

AD-A072 578

NAVAL POSTGRADUATE SCHOOL MONTEREY CA
A TACTICAL SYSTEM EMULATOR FOR A DISTRIBUTED MICRO-COMPUTER ARC--ETC(U)
JUN 79 L A GUILLEN

F/6 9/2

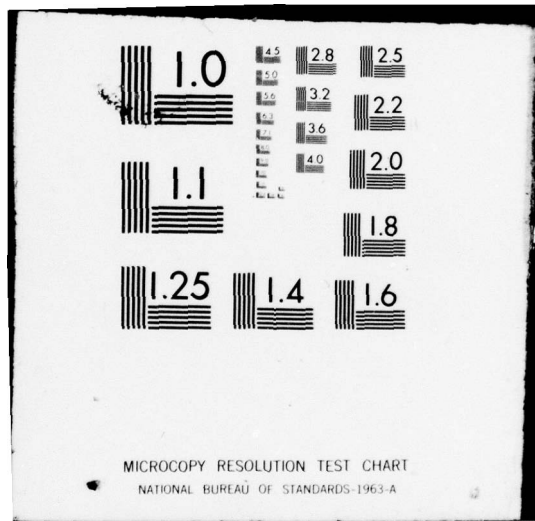
UNCLASSIFIED

NL

1 OF 4

AD
A072578





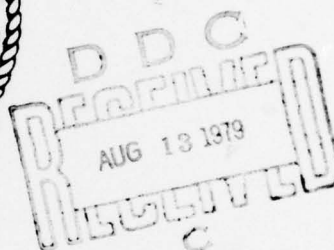
LEVEL II

2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD A 072578



THESIS

DDC FILE COPY

6 A TACTICAL SYSTEM EMULATOR
FOR
A DISTRIBUTED MICRO-COMPUTER ARCHITECTURE

by

10 Luis A./Guillen

11 June 1979

9 Master's thesis

Thesis Advisor:

Uno Kodres

Approved for public release; distribution unlimited

251 450

for

69 08 10 00 9

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Tactical System Emulator for a Distributed Micro-Computer Architecture		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1979
7. AUTHOR(s) Luis A. Guillen		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1979
		13. NUMBER OF PAGES 324
		15. SECURITY CLASS. (of this report) Unclassified
		16a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) EMULATOR, MICRO-COMPUTER, OPERATING SYSTEM, REAL-TIME SYSTEM, MULTIPROCESSING, TACTICAL SYSTEM		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) An emulator is constructed in order to study the behaviour of a Tactical System in a distributed micro-computer architecture environment. This emulator represents Tactical Systems as a set of periodic and demand scheduled functional module processes that collaborate with each other. A special purpose operating system was implemented for the distributed micro-computer architecture that supports the emu-		

lator. It includes processor managment and system wide input/output capabilities.

Accession For	
NTIS GNA&I	<input checked="checked" type="checkbox"/>
DDC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Availand/or special
A	

69 08 10 00 9

Approved for public release; distribution unlimited

A TACTICAL SYSTEM EMULATOR
FOR
A DISTRIBUTED MICRO-COMPUTER ARCHITECTURE

by


Luis A. Guillen
Lieutenant (JG), Peruvian Navy
B.S., Peruvian Naval Academy, 1974

Submitted in partial fulfillment of the
requirement for the degree of

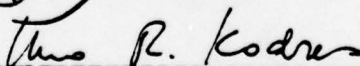
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
June 1979

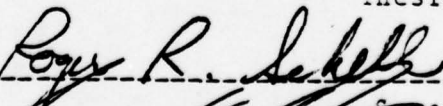
Author:



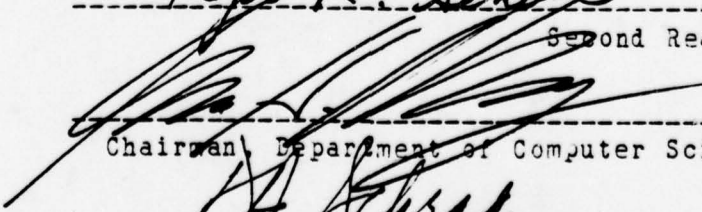
Approved by:



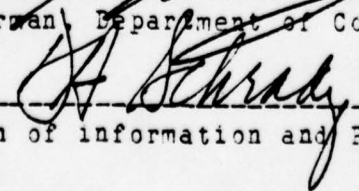
Thesis Advisor



Second Reader



Chairman, Department of Computer Science



Dean of Information and Policy Sciences

ABSTRACT

An emulator is constructed in order to study the behaviour of a Tactical System in a distributed micro-computer architecture environment. This emulator represents Tactical Systems as a set of periodic and demand scheduled functional module processes that collaborate with each other.

A special purpose operating system was implemented for the distributed micro-computer architecture that supports the emulator. It includes processor management and system wide input/output capabilities.

TABLE OF CONTENT

I.	INTRODUCTION	9
A.	MOTIVATION	9
B.	DESIGN OBJECTIVES	13
C.	THESIS BODY	15
II.	DISTRIBUTED SYSTEM	16
A.	GENERAL IDEA	16
B.	ARCHITECTURE	17
C.	THE EXECUTIVE	19
D.	INPUT/OUTPUT	21
E.	THE MONITOR	22
F.	USING THE SYSTEM	23
III.	DISTRIBUTED SYSTEM IMPLEMENTATION	25
A.	ENVIRONMENT	25
B.	SYSTEM INITIALIZATION	27
1.	First Step	28
2.	Second Step	29
3.	Third Step	30
C.	INTERRUPTS	33
1.	Hardware Characteristics	33
2.	Interrupt Configuration	36
3.	Interrupt Handler Routines	37
D.	COUNTERS	38
1.	Hardware Characteristics	38
2.	The Real Time Clock	40

3.	The Count Down Clock	41
4.	Baud Rate Generation	42
E.	SERIAL I/O INTERFACE	42
1.	Hardware Characteristics	43
2.	Serial I/O Priority Tasks	45
3.	Serial I/O Monitoring Procedures	46
4.	USART Programming	46
F.	PARALLEL OUTPUT INTERFACE	47
1.	Hardware Characteristics	47
2.	Parallel Output priority task	51
3.	Parallel Output Monitoring Procedures.....	52
4.	Peripheral Interface Device	52
G.	THE SYSTEM MONITOR	53
H.	SYSTEM'S ADDRESS SPACE	54
I.	PROGRAM DESCRIPTION	55
1.	File EXEC	56
2.	File INTMSG	58
3.	File SEIO	60
4.	File PAIO	61
5.	File MONI	62
6.	File SC1	62
7.	File SC2	63
8.	File SC3	64
9.	File SC4	66
10.	File SC5	66
11.	File SC6	68
12.	File SC7	70

13.	File MODPRO	70
14.	File RUN	70
IV.	TACTICAL SYSTEM EMULATOR	71
A.	DESIGN	71
1.	Dummy Functions	71
2.	Emulation Sequence	73
B.	IMPLEMENTATION	73
1.	Emulator-System Relationship	74
2.	Emulation Control	75
3.	Linking and Locating	76
C.	PROGRAM DESCRIPTION	76
1.	File EMULA1	76
2.	File EMULA2	77
3.	File EMULA3	80
4.	File EMULA4	81
V.	CONCLUSIONS	82
	APPENDIX A - SINGLE BOARD COMPUTER DESCRIPTION	84
	APPENDIX B - MULTIBUS DESCRIPTION	88
	APPENDIX C - HOW TO USE THE EMULATOR	90
	DISTRIBUTED SYSTEM PROGRAM LISTING.....	93
	EMULATOR PROGRAM LISTING.....	229
	LIST OF REFERENCES	323
	DISTRIBUTION LIST	324

ACKNOWLEDGEMENT

I must thank professor Uno Kodres, my advisor, for his support and constant dedication and professor Roger Schell for his comments and guidance. To them I owe most of the ideas of the thesis.

I also thank Mr. Bob McDonnell and Mr. Michael Williams, from whom I have learned so much during the implementation of the system. Finally I thank my wife, Marcela, for her patience.

I. INTRODUCTION.

A. MOTIVATION.

LSI technology has brought to the market new micro-computers which meet most of the requirements needed for tactical system development. They are highly reliable (hardware), small, easy to maintain, don't need much power, have a powerful instruction set and also, because of the great acceptance within the general market, they are relatively cheap and have considerable software support. The question is whether new architectures built from these computers will perform in a satisfactory manner within the real-time constraints imposed by tactical systems.

Tactical Systems are mostly dedicated systems, that is, they are designed to accomplish repetitive computations over the same algorithm. They are, also, functionally oriented, and most of the time the designer has a very good idea about the amount of execution time and memory space needed by each function before implementation takes place.

Tactical Systems can be viewed as a set of functional modules related by communication links. With this in mind, a tactical system could be mapped into a di-graph, where a node would represent a functional module and an arc the information crossing from one module to another.

As an example, take the di-graph in Figure 1. Nodes 1,2,3,4 and 5 represent functional modules which perform certain predetermined functions needed by the tactical

system. Arcs (a),(b),(c) and (d) represent information crossing from one module to another.

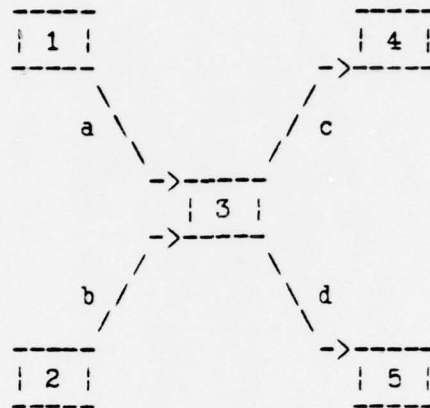


FIGURE 1. Example of a Di-graph

We must now observe that each node or set of nodes can be mapped into a process, where a process is characterized by an execution point and an address space, and each arc can be mapped into an interprocess communication message. With this mapping sequence we transform the original tactical system into a computer system design.

The task of partitioning the diagram so as to define the process set is not addressed in this thesis. Reference [4] reviews the published literature on graph partitioning.

Early tactical system implementations concentrate the computation effort in one relatively large, fast computer. In doing so, these implementations arrange the processing sequence into a totally ordered set. i.e. in our previous example, the processor would be used successively by

processes 1,2,3,4,5 and back to 1. The processor needs to be fast enough so it can be used by all the processes within the real-time constraints without diminishing the performance of the system. Sometimes these systems work based on a pooled algorithm in which the processes which do not need the immediate use of the processor are skipped until the next round. Even then, the processor must be such that it can handle the maximum load.

A different implementation approach can be taken. We could concentrate our effort in distributing the processes among several slower computers which collaborate with each other in pursuing the same system goals. Our main concern in distributing the processes among several computers is the resulting inefficiencies introduced by the distribution:

- 1) Greater communication problems between computers.
- 2) Duplication of data and programs.
- 3) Imbalance in execution time and program size among the processors.

Reference [3] explains a data flowgraph technique that allows us to explicitly determine the number of data elements which must be communicated between processes. The flowgraph analysis allows us to conclude if our processes could be carried out in different computers completely independent without creating communication problems.

Given that we have followed the last approach, we are interested in finding out whether we can meet the real-time requirements imposed by the tactical systems. Following our

previous example, we can examine the idea using Figure 2.

CPU A is the fast processor used in the centralized implementation. Assume we have two slower processors (CPU 1 and CPU 2) for our distributed approach. Given that we have determined that processes 1 and 2 and then 4 and 5 can be executed , we want to find if the time lapse in which the five processes are executed by CPU 1 and CPU 2 meet the real-time requirements.

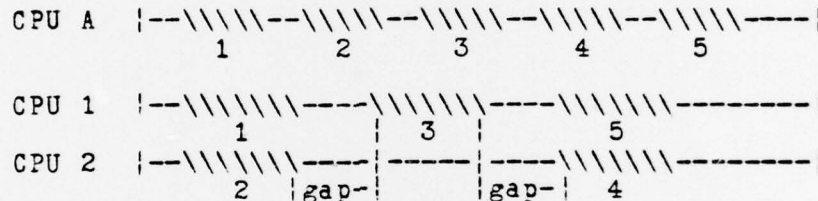


FIGURE 2. Time lines for centralized and distributed approaches.

In most typical tactical systems, we can estimate the execution time required by each process, as was mentioned before. The critical point in finding the time frame in which the system is going to work is the interprocess execution gaps and how much they diminish the overall performance.

A hardware architecture has been proposed in Reference [3]. In this architecture the system is built from identical single board processors such as the INTEL SBC80-20 or the Texas Instruments TM 9900. The boards are connected by the INTEL MULTIBUS, TI TILINE or the Digital Equipment's UNIBUS. A group of single board computers connected on a parallel bus is called an 'affinity group'.

A Real-Time Operating System for Single Board Computer Based Distributed Naval Tactical Data System was developed at the Naval Postgraduate School (see Reference [2]) in order to support the implementation of tactical systems, on an architecture of independently operating single board processors.

In order to determine if such architectures can effectively support tactical processing, the results of analysis should be verified by a more realistic emulation of the system. It is the purpose of this thesis to implement an emulation system based on a multiple single board computer architecture and the real-time distributed operating system [Ref. 2].

B. DESIGN OBJECTIVES.

As mentioned before, a Tactical System could be mapped into a system of independent processes related by interprocess communication messages. The goal of this thesis is to create a Tactical System Emulator in order to study the behaviour of such a system in a distributed microcomputer architecture.

The Tactical System Emulator intends to be a tool for helping the tactical system designer in allocating the different processes, into which his system has been partitioned, between the processors available in a proposed microcomputer network. By using the emulator, the system's designer can identify potential communication bottlenecks

before the system is fully implemented.

The Tactical System is viewed for this purpose as a set of functional modules that interact with each other through messages. There are two kinds of functional modules: periodic and demand. Periodic functional modules become active every predetermined interval of time. They are triggered by the system and are usually in charge of input/output functions. Demand functional modules become active upon receipt of a message. This message can be sent by periodic as well as by demand functional modules.

The Emulator replaces every periodic functional module with a dummy periodic module and every demand functional module with a dummy demand module. These dummy modules are not expected to perform the required function but to consume execution time in the same way as if they were doing it. They are also expected to transmit dummy messages of the same length and destination as the ones sent by the real functional modules they are replacing.

At execution time, the Emulator would save statistics about the behaviour of the system with emphasis on the interprocess communication delays.

The emulator would also provide a dynamic tactical system's definition tool. This definition tool will allow the user to define the real Tactical System's parameters and then will distribute this information among the Single board computers, so as to prepare the emulator data structure for the emulation. In other words, the user will interact with

the emulator system in preparing the emulation environment. The emulator will require system parameters such as the CPU time needed by each functional module, number of messages , destination and length of each message, number of processors needed and so on.

It would be also necessary that the emulator allow the user to redistribute the modules between processors and to modify the emulation parameters dynamically.

C. THESIS BODY.

The architecture of the distributed system of SBC80-20 micro-computers was chosen because the availability of the single board computers in the micro-computer laboratory. The availability of the system's programming language, PLM80, and the existence of a previous thesis which implements a real-time operating system for the same architecture (see Reference [2]) were equally important reasons for choosing this architecture.

Tailoring this operating system to meet the emulator needs and to run in the choosen architecture, took most of the thesis effort. Chapter II and III explain the characteristics of the system created to support the emulator and the implementation details.

Chapter IV describes the emulator as it was conceived and implemented. Appendix C explains how to use the emulator.

II. DISTRIBUTED SYSTEM.

A. GENERAL IDEA.

Real-time applications push computer and programming technology to its limits (and sometime beyond). A real-time system is expected to monitor simultaneous activities with critical timing constraints continuously and reliably. The consequences of system failure can be serious.

Real-time systems must achieve the ultimate in simplicity, reliability, and efficiency. Otherwise one can neither understand them, depend on them, nor expect them to keep pace with their environment.

To make a real-time system efficient will probably require the design of computer architectures tailored to particular applications. Real-time systems have these characteristics:

(1) A Real-time System interacts with an environment in which many things happen simultaneously at high speeds.

(2) A Real-time system must respond to a variety of asynchronous requests from its environment. The system can't predict the order in which these request will be made but must respond to them within certain time limits. Otherwise, input data may be lost or output data may lose its significance.

(3) A Real-time System controls a computer with a fixed configuration of processors and peripherals and performs (in most cases) a fixed number of concurrent task in its

environment.

(4) A Real-time System never terminates but continues to serve its environment as long as the computer works. (The occasional need to stop a real-time system, at the end of an experiment can be handle by an ad hoc mechanism, such as turning the machine off or loading another system into it.)

What is needed then for real-time applications is the ability to specify a fixed number of concurrent tasks that can respond rapidly to asynchronous requests. The system proposed here is a real-time system with the following characteristics:

(1) A Real-time System consist of a fixed number of concurrent processes that are started simultaneously and exist forever. Each process can access its own variables as well as shared information.

(2) Processes can interchange information by sending messages to each other. This way a process can request services from another process. This is the only form of interprocess communication.

(3) Processes are synchronized by means of interprocess communication messages.

B. ARCHITECTURE.

The proposed architecture consist of a set of identical single board computers and read/write memory boards connected by a time multiplexed data bus.(see Figure 3.)

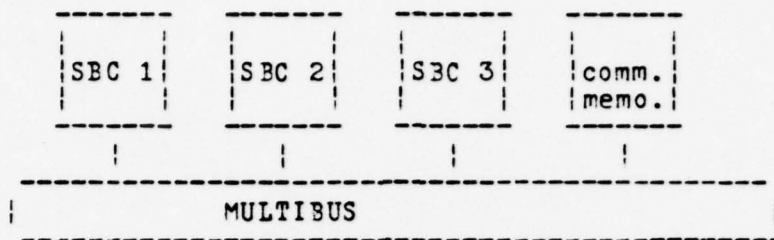


FIGURE 3. An "Affinity Group".

Each Single Board Computer (from now on named SBC) is able to communicate with the others by means of a message buffer located in common memory (the memory boards). When one SBC wants to transmit some information to another, it would store a message with the information in common memory and then the receiver will pick it up from there.

The system to be implemented could be defined as a special purpose operating system. It provides interprocess communication, in the form of message routing, and process scheduling. It also provides system wide input/output facilities. Memory and Information management are not considered necessary for this class of tactical operating system, so these functions must be provided by the user.

The Operating System recognizes three kinds of tasks: Priority task, Message task, and Periodic task. All user's applications must be tailored to fit into these categories.

Priority task are scheduled for system events. These usually cover the input/output and real-time clock computational needs. A zero count in a system count-down clock, an interrupt or a system reset are examples of event that would produce the scheduling of one of these task.

Message tasks are scheduled upon receipt of a message for them. These are usually defined by the user and carry most of the computations.

Periodic tasks are scheduled at the end of a time-lapse defined by its period. A periodic task is defined with an entry point, and an interval of time (period). The system keeps track of the next activation-time and the state of the task.

C. THE EXECUTIVE.

Each SBC is driven by an executive which resolves priorities between system task based on the algorithm described by the flowchart in Figure 4.

Priority tasks have the highest priority. Message tasks have the next highest priority. Periodic tasks have the next highest priority and background tasks proceed at the lowest priority level.

The Executive must check for messages coming from two different sources. One is the SBC buffer used by the system as an immediate message buffer. The other is the System buffer used as a system wide message buffer. Messages directed to messages task allocated in the same computer as the source will only use the SBC buffer. If the directed message task is not allocated in the same computer the message will be put in the system buffer to be picked up by the respective executive.

Periodic task initiation is based on the content of the

real-time system clock. Each periodic task is associated with an activation time. When this time is lower or equal than the real-time clock the task will be scheduled. The real-time clock must be system wide so that the reference for all the executives will be the same.

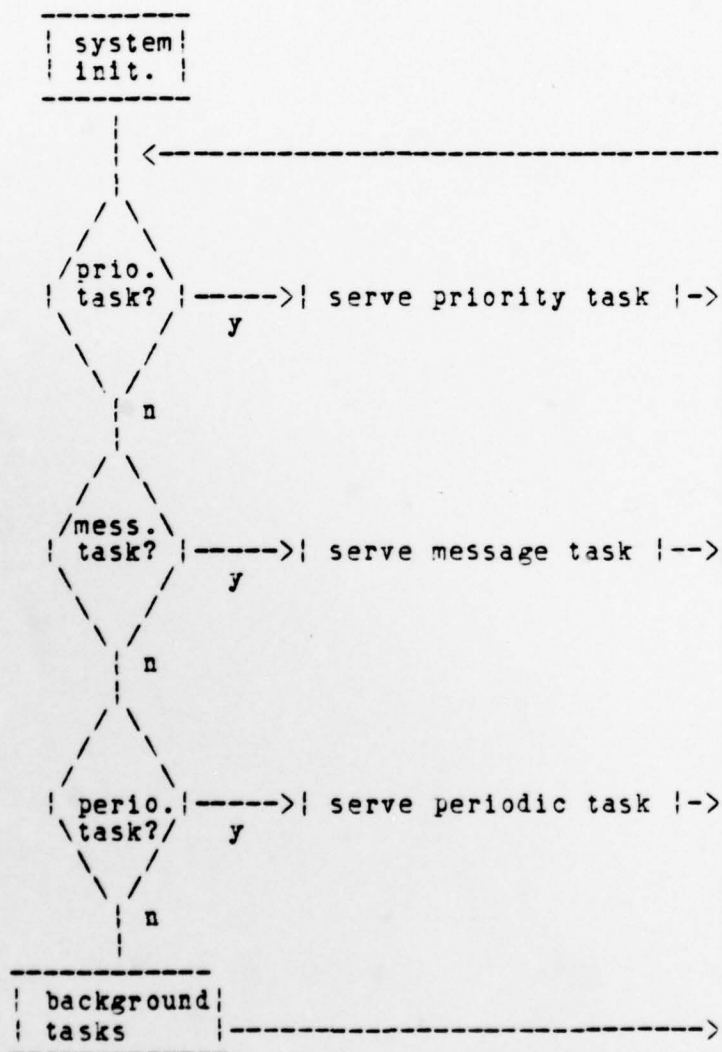


FIGURE 4. Executive Flowchart.

D. INPUT/OUTPUT.

Each SBC must be able to handle its own input/output resources. The presence of data to be transmitted or received from an external device will be considered as a system event able to be identified locally by the SBC to which the device is connected.

This input/output event will produce the scheduling of a corresponding priority task which would be able to receive or transmit the data.

All input/output procedures then would be made at this level. The interchange of data will be between the external devices and locally allocated input/output buffers.

A combination of priority and message tasks should be used for serving the interrupt driven input/output functions. Interrupt driven I/O must be used rather than dedicated I/O in order to allow other tasks to be served while the communication interchange is in progress and contributed to meet the real-time constraints as needed.

For the serial I/O interface constructed, an input (priority) task will collect the input data in a buffer. When the input task receives an end of data notification, it will empty the buffer by sending a message to the task which uses the data.

A virtual system console, able to be attached to any one of the processors is created. It provides the user with a powerful tool for constructing user interface software. Ad hoc system routines like PRINT\$SYNC and CON\$INPUT\$REQ, for

input/output to this console, are offered to the user in order to facilitate the control of the virtual device.

Because the characteristics of the system, an input operation is not a function but a request for an input message string. In the case of the system console, the message will come from the real device attached to it.

D. THE MONITOR.

Special software was created to allow the user to interact with the system in order to examine and modify its parameters.

As defined before, each SBC has its own local memory which the system uses to store data and certain parameters. In the development of applications it is very important to have access to the data not only for debugging purposes but to be able to recognize transient states that can help in the improvement of the application.

The system monitor contains commands for displaying, filling, and modifying memory, for the transmission of messages from the console and the allocation of this console to any one of the computers. It interacts with the input/output task for communicating with the user and is mainly composed of message tasks.

A special feature includes triggering the monitor when control gets to a defined execution point. This event, requested by the monitor with a command, will schedule a priority task which in turn will schedule the monitor of the

computer which got to this execution point.

E. USING THE SYSTEM.

This special purpose executive operating system has not been created with the idea of a stand alone system but as a tool that must be used in conjunction with the program development system that helps the user to set up the initial state.

Each SBC will have in its read only memory a copy of the executive, a serial input/output handler, a parallel output handler, an interrupt handler, a monitor and a set of system routines that will help the user to control the on-board system resources and the communication between processes.

The system data structures include an address vector for priority task entry points, an address vector for message task entry points and status, and an information vector for each periodic task containing status, entry point address, interval time address and next activation time. This data, in conjunction with a real-time clock, a message buffer and other system parameters, which will be described in Chapter III are allocated in common memory. The system provides system routines for controlling these data structures.

Message tasks are created by defining a message entry point and an initial status (active or inactive). This can be done before the distributed system initialization, provided the initialization message is accepted. Any periodic task can be created, suspended and its period

modified using system routines. In the same way priority task can be controled.

In addition of the purposes mentioned previously, a variety of system routines have been created for helping the tactical system designer. The user would only need to properly link his programs with the system's public reference in order to use them.

Special care must be taked in the linking and locating process at implementation time, because there is no mechanism for memory management. The user himself would be responsible for the proper location of his programs.

Each SBC can control (essentially by multiprogramming) up to eight priority tasks, eight message tasks, and eight periodic tasks.

III. DISTRIBUTED SYSTEM IMPLEMENTATION.

A. ENVIRONMENT.

The system has been built from three INTEL SBC80/20-4 Single Board computers and four 16K RAM random access memory boards. The description of the SBC and the MULTIBUS by which these boards communicate with each other can be found in Appendix A and B respectively. Detailed information about the computer is found in Reference [5].

In order to program the software for the system, the INTELLEC MDS Microcomputer Development System was used. A complete description of this system can be found in Reference [6].

Software used for the development of the distributed system, as well as for the emulator, include ISIS-II Operating System and PLM80 compiler. Information about these software products can be found in References [7,8].

The distributed system's memory organization can be seen with the help of Figure 5.

- ISIS-II operating system uses the lowest 12K bytes of common memory. Locations from 0000H to 2FFFH.

- MDS monitor uses the highest 2K bytes of common memory. Locations from F800H to FFFFH.

- The distributed system code uses the lowest 3K bytes of on board memory, locations from 0000H to 1FFFH. This memory segment contains the Executive, interrupt handler, messages handler and input/output handlers, the system

initialization software, the monitor and all the system routines. One copy for each single board computer.

- The Distributed System interrupt vectors and local variables use on board SBC memory from 3000H to 38FFH. On board memory from 3900H to 3FFFFH is available to the user.

- The Distributed System general data structures are allocated in common memory from F400H to F7FFH.

- Locations from 4000H to F3FFH are free for use. These are 45K bytes of common memory available to the user.

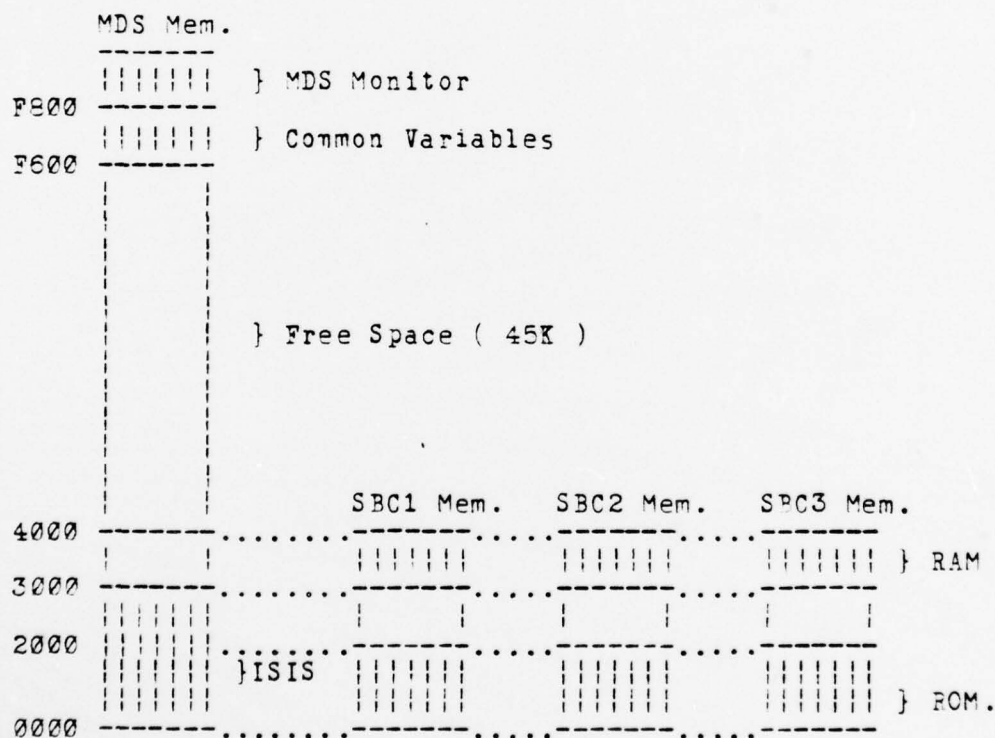


FIGURE 5. System's Memory Organization

On board SBC memory shadows the corresponding locations on common memory, so that the SBC can't access common memory from 0000H to 1FFFH or from 3000H to 3FFFFH. In the same way,

a SBC's is protected against other processors trying to access its local memory.

The MDS System provides a front panel board which, besides working as an interface between the front panel switches and the MDS processor and other functions unimportant in this context, serves as a priority resolver for the Bus priority logic. It resolves bus contention for up to eight master modules. The logic monitors eight bus request (BREQ) lines, arbitrates all request in parallel, and controls eight bus priority enable (BPRN) lines. Only one BPRN line associated with the highest priority module which is requesting use of the bus is enabled, thus allowing that board to become bus master. The front panel board also generates the 9.8 MHz bus clock (BCLK/) signal which provides a timing reference for the bus control section of the various master modules. This board is the only other board used after the Distributed System has been initialized. Another feature of the front panel board, namely the interrupt switch logic for interrupt number six is used as a "warm boot" for reinitializing the system (without turning the power off).

The ISIS-II Operating System is used to develop the programs for the Distributed System applications, as was the case with the Emulator programs.

B. SYSTEM INITIALIZATION.

Several steps must be taken before the executive takes

control of the system. They can be separated into three steps.

The first one is executed under ISIS-II Operating System, working with the standard MDS processor and system console; its purpose is to link the user's application programs within the Distributed System.

The second step will put the MDS processor into a HALT state and will activate the SBC's which are needed as part of the system.

The third step will be taken by each SBC, provided it has been activated, and its purpose is to prepare the local data structures and resources for the new job.

1. First Step

The user can make use of several system routines, which will be described in detail later. In order to do so, he only needs to declare them external variables and then reference them as indicated. Remember that the code belonging to these routines is already present and has been allocated in on board SBC memory. Because of the characteristics of the hardware, the memory cycle in this memory is free from bus contention and hence the use of these routines is highly recommended over similar ones created by the user.

To get to the system routines, it would be sufficient for the user to link his previously compiled code with the public reference of the Distributed System code.

This would resolve any external reference and would direct the processor to the corresponding routine in local memory.

The user must be sure that the initial state of the message task vector address is as he wishes. The message task vector address is one of the system wide data structures located in common memory. Its name is MSG\$MOD\$ADDRESS and is located at MSG\$TEL\$ADR (F660H).

The program MOD\$PRO.SRC provides two procedures for initializing this structure: CLEAR\$MOD, which erases the entire vector and ENTER\$MOD(#, Address), which initializes a message task by defining the message number to be used and the entry point address of the task. Computer 1 serves message tasks numbered between 1 and 7; computer 2, message tasks numbered between 9 and 15 and computer 3, message tasks numbered between 17 and 23. Numbers 3, 8 and 16 are reserved for the Distributed System itself.

2. Second Step

Upon system reset, all three SEC's begin to look at a special system wide variable called LOADSBC. When LOADSBC takes the value equal to the number with which the SEC identifies itself, it is the time for waking up. First thing to do for the SEC would be to check a second vector variable called START in the position corresponding to its identification number (1,2 or 3). If the computer can find the same number in this position, then it proceeds to initialize itself, otherwise it keeps checking START every

second, forever.

The program called RUN, which executes under ISIS-II, receives as input the number of the SBC's which the user is willing to use. SBC's are identified by the numbers 1, 2 and 3. Upon receipt of the input, it sets the corresponding positions of START and sets variable LOADSBC to 1. LOADSBC will be incremented by computer 1 and then by computer 2, in case it has been requested. The program then puts the MDS processor in HALT state.

The program also provides for a "soft boot" reset routine with interrupt number six. This interrupt would be used as a reinitilization interrupt by the Distributed System.

3. Third Step

Several steps must be taken by each SBC before giving control to the executive. They include the initialization of:

- (a) system wide data structures.
- (b) executive parameters.
- (c) interrupt controller.
- (d) counters.
- (e) serial and parallel I/O interfaces.
- (f) interrupt mask.

We should see these steps in more detail.

a. System data structures initialization.

Before the executive takes control, the

following must be accomplished with the data:

(a) System wide message buffer, EXTMSGBUFFER must be empty and the corresponding pointers and flags set to zero.

(b) The real-time clock RTC set to zero time, and its semaphore, CLOCK set to free.

(c) Immediate message buffer, MSGBUFFER must be empty and the corresponding pointers and flags set to zero.

(d) The address vector for priority task and periodic task (PRIORLIST and PERLIST) set to null.

(e) MSGENT0 must be entered as a message task in the corresponding number (0, 8 or 16). This task groups the entry points of the I/O interfaces and the monitor.

(f) All control parameters and variables set to zero.

All the step listed above are accomplished by procedure SET\$EX\$DATA in INTMSG.

b. Executive parameters initialization.

The system provides an initialization message, numbered 0, and several system messages for controlling the system console and the printer. Initially, the system console and printer will be defined in SEC 1, so these messages will be directed to message task number 0. These messages are numbered from 10 to 25 and are initialized by procedure SET\$EX\$MSGs in INTMSG.

The procedure SET\$EX\$MSGs will also allocate an initialization message for each user message task defined

for the SBC.

c. Interrupt controller initialization.

When the processor receives an interrupt signal it checks the location of the corresponding interrupt handler in an interrupt vector previously defined. These interrupt vectors must be initialized and their location and characteristics communicated to the interrupt controller hardware. This is done by procedure SET\$EX\$INTE in INTMSG.

d. Counters initialization.

The INTEL SBC80/20 hardware has three counters, numbered 0, 1 and 2. Counter 0 is used to update the real-time clock, RTC. Counter 1 is used for a special mechanism which will trigger a user defined procedure at the end of the required count. Counter 2 is used for baud rate generation for serial I/O communication interface.

All three counters must be programmed to accomplish the purpose they have been assigned to do before they are used.

e. Input/Output interface initialization.

Serial and Parallel I/O interfaces are initialized by procedures SEIO\$START and PAIO\$START respectively. In general they must set the hardware for the proper communication protocol, clear the I/O buffers and pointers, and introduce the corresponding I/O priority task into PRIORLIST.

f. Interrupt mask initialization.

The interrupt mask is saved in system variable

INTMASK. In case of computer 1, the mask would be set to allow counter 0 to produce an interrupt for RTC updating, and to enable interrupt six in order to reinitialize the initialization sequence whenever the user wants to push the front panel switch. Besides this, all SBC interrupts three, four and five are enabled for I/O purposes.

Counters, I/O interface, and interrupt mask initialization is accomplished by procedure EXSTART in INTMSG.

C. INTERRUPTS.

1. Hardware characteristics.

The INTEL 8259/20-4 interrupt controller logic consist of Intel's 8259 Interrupt controller device and a jumper pad that allows the user to connect any of 27 possible interrupt request to the 8259's eight interrupt priority inputs. The 8259 resolves priority among all eight levels according to an algorithm which is program selected by the user. The normal interaction of the 8259 with the CPU is as follows:

(a) one or more of the interrupt request lines (IR7-0) is raised high signalling to the 8259 that the peripheral equipment is demanding service.

(b) The 8259 accepts these requests, resolves the priorities, and sends an INT to the 8080 CPU.

(c) The 8080 CPU acknowledges the INT and responds

with an INTA/ pulse.

(d) Upon receiving the INTA/ from the CPU group (8238), the 8259 will release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7-0 pins.

(e) This CALL instruction will initiate two more INTA/ pulses to be sent to the 8259 from the CPU group (8238).

(f) These two INTA/ pulses allow the 8259 to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first INTA/ pulse and the higher 8-bit address is released at the second INTA/ pulse.

(g) This completes the three byte CALL instruction released by the 8259. The in-service register (ISR) is not reset until the end of the subroutine when an EOI (End of interrupt) command is issued to the 8259.

The 8259 accepts two types of command words generated by the CPU for programming purpose:

a. Initialization Command Words (ICWs):

Before normal operation can begin, each 8259 in the system must be brought to a starting point--by a sequence of 2 or 3 byte commands timed by WR/ pulses.

b. Operation Command Words (OCWs):

These are the command words which command the 8259 to operate in various interrupt modes. These modes are: Fully nested mode, Rotating priority mode, Specific priority mode and polled mode.

The 8259 will operate in the fully nested mode after the execution of the initialization sequence without any OCW being written. In this mode, the interrupt requests are ordered in priorities from 0 to 7. When an interrupt is acknowledged, the highest priority request is determined and its address vector placed on the bus.

Whenever a command is issued with $A0 = 0$ and $D4 = 1$ this is interpreted as initialization command word 1 (ICW1), and this initiates the initialization sequence. During this sequence, the following occurs automatically:

(a) The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low to high transition to generate an interrupt.

(b) The interrupt mask register is cleared.

(c) IR 7 input is assigned priority 7.

(d) The special mask mode flip-flop and the status read flip-flop are reset.

Initialization command word 2 must be output right after ICW1. ICW1 provides two control bits and two or three of the interrupt CALL address bits. ICW2 provides 8 of the CALL address bits.

The 8 requesting devices have 8 addresses equally spaced in memory. The addresses can be programmed at intervals of 4 or 8 bytes: the 8 starting locations therefore occupy 32 or 64 bytes of memory respectively.

2. Interrupt Configuration.

There are two major consideration in configuring the interrupt structure on the SBC 80/20:

(a) The connection of external and on-board interrupt requests to the eight interrupt priority level inputs (IR0-IR7) on the 8259.

(b) The selection of a priority resolution algorithm.

The priority resolution algorithm is selected by programming the 8259 as mentioned before. The assignment of interrupt requests to interrupt priority level inputs is made by connecting the appropriate jumper pins.

The 8259 was programmed in the fully nested mode for the Distributed System. Interrupt 0 has the highest priority and interrupt 7 the lowest.

The interrupt section jumper pad was connected in the following way:

- (a) Interrupt 0 (pin 24) with (pin 34).
- (b) interrupt 1 (pin 25) unconnected.
- (c) Interrupt 2 (pin 26) with (pin 35).
- (d) Interrupt 3 (pin 27) with (pin 41)
- (e) Interrupt 4 (pin 28) with (pin 40).
- (f) Interrupt 5 (pin 29) with (pin 63).
- (g) Interrupt 6 (pin 30) with (pin 49).
- (h) Interrupt 7 Unused.

3. Interrupt Handler routines.

Interrupt 0 was used to handle a special system event associated with Counter 1. Using system routine SETCDC(lapse,address) the user can request the system to execute the procedure whose address is indicated after the time lapse he gave. This system routine will set counter 1, which acts as a count down counter, with the specified time lapse. Counter 1, which is attached to interrupt 0, will produce an interrupt when it gets to zero, and the interrupt handler will call the procedure as required. This mechanism must not be used frequently, nor should the procedure consume too much processor time.

Interrupt 1 was used to handle the monitor trap. The user can, by a command, activate the monitor when the processor gets to a certain execution point. To accomplish this, the monitor replaces the code at that point by an interrupt 1 command code. Whichever processor gets to that point first would execute the interrupt 1 instruction and would then get to the MONITOR\$TRAP procedure.

Interrupt 2 was used to update the real-time clock.

Interrupt 3 was used to schedule the serial input priority task. It is triggered by the Receiver Ready signal in the USART which goes high when a character is received from the keyboard.

Interrupt 4 was used to schedule the serial output priority task. It is triggered by the transmitter ready signal in the USART, which goes high when the USART buffer

has been emptied.

Interrupt 5 was used to schedule the parallel output priority task. It is triggered when a ready to receive signal is received from the printer.

Interrupt 6 was used for system reinitialization. It is connected to the front panel interrupt 6 switch.

D. COUNTERS.

1. Hardware Characteristics.

The SBC 80/20 includes an 8253 Programmable Interval Timer. The 8253 solves one of the most common problems in a micro-computer system, the generation of accurate time delays under software control. Instead of setting up timing loops in system software, the programmer configures the 8253 to match his requirements, initializes one of the counters of the 8253 with the desired quantity, then upon command the 8253 will count out the delay and interrupt the CPU when it has completed its task. It is easy to see that the software overhead is minimal and that multiple delays can easily be maintained.

Other counter/timer functions that are non-delay in nature but also common to most micro-computers can be implemented with the 8253.

- Programmable Baud Rate Generator
- Event Counter
- Binary Rate Multiplier

- Real Time Clock
- Programmable One-Shot
- Complex Motor Controller

The 8253 includes three separate counters. Each counter is a single 16-bit pre-settable down counter. Each counter can operate either in binary or in BCD and its outputs are configured by the selections of function stored in the Control Word Register and jumper configurations on the SBC 80/20.

The counters are fully independent and each can have separate mode configuration and counting operation, binary or BCD. Also, there are special features in the Control Word that handle the loading of the count value so that software overhead can be minimized for these functions.

The complete function definition of the 8253 is programmed by the system software. A set of control words must be sent out by the CPU to initialize each counter of the 8253 with the desired function and quantity information. These control words program the function, loading sequence and selection of binary or BCD counting. Once programmed, the 8253 is ready to perform whatever timing task it is assigned to accomplish.

a. Mode 0: Interrupt on Terminal Count

The output of the 8253 counter will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low but the counter starts to count. When terminal count is

reached, the output will go high and remain high until the selected count register is reloaded, or the mode set again.

Reloading a counter during count will restart the procedure. A gate input will enable counting when high and inhibit counting when low.

b. Mode 2: Rate Generator

Divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next is equal to the number of input counts in the count register. If the count register is reloaded between output pulses, the present period will not be affected, but the subsequent period will reflect the new value.

The Gate/Reset input, when low, will force the output high. When the Gate/Reset input goes high, the counter will start from the initial count. Thus, Gate/Reset input can be used to synchronize the counter.

2. The Real Time Clock

The Distributed System Real Time Clock is implemented using Counter 1 of the single board computer number one. The output from Counter 1 is attached to interrupt input pin number two and the count register loaded with a value that will produce an interrupt after one millisecond. The interrupt handler will increment the real time clock vector RTC and will reload the value into the count register to produce the next interrupt.

Counter 1, as all the counters in the 8253, counts at a rate of 930 nano-second. In order to produce an interrupt after one millisecond the count register must be loaded with a value close to $1000000\text{ns} / 930\text{ns} = 1075.2688$. It was chosen $1075 = 433\text{H}$.

SEC 1 is the only one whose interrupt mask is programmed to enable interrupt 2. It will update the real-time clock. Procedure INT2 in INTMSG accomplishes the update. The procedure checks for the variable, CLOCK, to be free before modifying the clock. This prevents it from disturbing any procedure which is reading the real time clock.

3. The Count Down Clock

The Count Down Clock is a feature implemented using Counter 0 from any SBC. This counter is programmed in mode 0, in the same way as Counter 1 for the real time clock. The user can request any processor to execute the indicated procedure as well as determine the count value. The system will load this value into the count register of Counter 0 and will enable interrupt 0, to which the counter is attached, by modifying the interrupt mask. The procedure entry point address is saved in variable CDCADR and a flag variable CDCACTIVE is implemented to ensure no side effect errors.

The user requests this service using system routine SETCDC(value,address) The value can not exceed FFFFH and the

user must remember that the rate is 930 nanoseconds. The procedure will be called by the interrupt handler INT0 in INTMSG.

4. Baud Rate Generation

Counter 2 has a dedicated function on the SBC 80/20. This counter provides a baud rate clock for to the serial I/O interface. The output from Counter 2 is made available to the 8251 USART where it can be jumpered to the Receiver and/or Transmitter clock inputs. Counter 2 is always enable (its gate input is tied to +5v).

Counter 2 must be programmed in mode 3 to generate the baud rate clock. As implemented, it generates a baud rate of 2400 bits per second.

E. SERIAL INPUT/OUTPUT INTERFACE

Each SBC would be able to handle a peripheral device connected to it by its J3 connector. This is a serial RS232 interface connector associated with the Universal Synchronous/Asynchronous Receiver/Transmitter (USART) communication interface. Receiver Ready (RxRDY) and Transmitter Ready (TxRDY) signal from the USART were wired to the interrupt controlled (interrupt 3 and 4) so the input/output functions would be performed by the priority task scheduled by the interrupt handlers 3 and 4. A set of procedures commanded from the system message task (number 0, 8 or 16) were implemented in order to monitor the operation

of these priority task.

1. Hardware Characteristics

The serial I/O interface logic provides the SBC 80/20 with a serial data communication channel that can be programmed to operate with most of the current serial data transmission protocols, synchronous or asynchronous. Baud rate, character length, number of stop bits and even/odd parity are program selected.

The serial I/O interface logic consist primary of an Intel 8251 USART device and several driver/receiver circuits.

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control words define the complete functional definition of the 8251 and must immediately follow a system reset operation (internal or external).

The control words are split into two formats:

- Mode Instruction
- Command Instruction.

Both the mode instructions and the command instruction must be given in a specified sequence for proper operation. The mode instruction must be inserted immediately following a reset operation, prior to using the 8251 for data communication.

All control words loaded into 8251 after the mode

instruction will load the command instruction. Command instructions can be written into the 8251 at any time in the data block during the operation of the 8251. To return to the mode instruction format a bit in the command instruction word must be set to initiate an internal reset operation which automatically places the 8251 back into the Mode Instruction format.

a. Mode Instruction:

This format defines the general operational characteristic of the 8251. It must follow a reset operation. Once the mode has been written into the 8251 by the CPU, a SYNC character or command instructions can be inserted.

The 8251 can be used for either synchronous or asynchronous communication. The two least significant bits of the mode instruction control word specify the kind of operation.

b. Command Instruction:

Once the functional definition of the 8251 has been programmed by the mode instruction and the sync characters are loaded (if in sync mode), then the device is ready to be used for data communication. The command instruction controls the actual operation of the selected format. Functions such as: Enable Transmit/Receive, Error Reset and Modem controls are provided by the command instruction.

c. Data Transfers:

Once programmed, the 8251 is ready to perform its communication functions. The TxRDY output is raised 'high' to signal the CPU that the 8251 is ready to receive a character. This output (TxRDY) is reset automatically when the CPU writes a character into the 8251. On the other hand, the 8251 receives serial data from the modem or I/O device; upon receipt of an entire character, the RxRDY output is raised "high" to signal the CPU that the 8251 has a complete character ready for the CPU to fetch. RxRDY is reset automatically after the CPU read operation.

The 8251 cannot begin transmission until the TxEN (Transmitter Enable) bit is set in the command instruction and it has received a clear to send (CTS) input. The TxD output will be held in the marking state upon reset.

2. Serial I/O Priority Task

The actual character I/O functions are performed by a priority task, scheduled by the interrupt handler. Procedure CSINPUT in SEIO is the serial input priority task and CSOUTPUT in SEIO is the serial output priority task; they were defined as such in procedure SEIO\$START in SEIO during system initialization.

CSINPUT puts the incoming characters into the serial input buffer CRTIN and also into the serial output buffer CRTOUT for echoing. Upon receipt of a carriage return character (CR), CSINPUT checks if the last string was requested by the connected module, in which case it will

sent the character string to the requesting module.

CSINPUT also identifies a special character defined by MONITOR\$ID; when this character is sensed, the system routine MONITOR will be called which will activate the System Monitor.

CSOUTPUT dumps the serial output buffer CRTOUT one character at a time. It also senses if the connected module wants an acknowledge message when the buffer is empty, in which case CSOUTPUT sends the acknowledge message.

3. Serial I/O Monitoring Procedures

All requests for the use of the serial I/O priority tasks are sent to the System Message Tasks. The requests will be directed to message task 0, for computer 1; message task 8, for computer 2; and message task 16, for computer 3. These message tasks controls a set of procedures that keep track of the module connected to the facility.

Before any input string is passed, the connected task requesting for it is checked. Only a connected task can deposit data into the serial output buffer. The priority task CSOUTPUT would not be scheduled if the TxEN (Transmit Enable) bit in the USART command word were not set, and only the connected task has access to it.

4. USART Programming

The USART is programmed in procedure INIT\$USART, called from SEIO\$START in SEIO during system initialization.

It is set for asynchronous communication at 2400 bits per second. Formatting details can be found in Reference [5].

F. PARALLEL OUTPUT INTERFAS

The distributed system provides also for parallel output devices to be connected to the single board computer. It makes full use of the hardware facilities and programmable device characteristics of the SBC 80/20. The acknowledge signal coming from the peripheral device after it accepts a sent character has been directed to the interrupt controller and connected to interrupt input pin number 5. In this way, once one character from the output string has been acknowledge, the subsequent priority task for sending the remaining characters in the string will be scheduled by the interrupt handler number 5.

1. Hardware Characteristics

The parallel I/O interface logic in the SBC 80/20 provides forty-eight (48) signal lines for the transfer and the control of data to or from peripheral devices. Sixteen lines have a bidirectional driver and termination networks permanently installed. The remaining thirty two lines are uncommitted. Sockets are provided for the installation of active driver networks or passive termination networks. The optional drivers and terminators are installed in groups of four by insertion into the 14-pin sockets.

All forty eight signal lines emanate from the I/O

ports on two Intel 8255 Programmable Peripheral Interface devices. The two 8255 devices allow for a variety of I/O configurations.

a. Operational Summary

The 8255 contains three 8-bit ports (A, B, and C). Each Port can be configured in a wide variety of functional forms by the system's software.

(1) Port A: One 8-bit data output latch/buffer and one 8-bit data input latch.

(2) Port B: One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

(3) Port C: One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under mode control. Each 4-bit nibble contains a 4-bit latch and it can be used for control signal outputs and status signal inputs in conjunction with Ports A and B.

The 8080 CPU controls the operating characteristics of the ports by sending two different types of control words to the 8255:

- 1) mode definition control word
- 2) Port C bit set/reset control word

Bit seven of each control word specified its format.

b. Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0 - Basic Input/Output

Mode 1 - Strobed Input/Output

Mode 2 - Bi-directional Bus

When the RESET input goes 'high', all ports will be set to Input mode 0 (i.e. all 24 lines will be in the high impedance state). After the RESET is removed the 8255 can remain in the input mode with no additional initialization required. During the execution of the system program, the other modes can be selected using a single output instruction. This allows a single 8255 to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed.

c. Single Bit Set/Reset Feature

Any of the bits of Port C can be set or reset with one output instruction. This feature reduces software requirements in control based applications.

When Port C is being used as status/control for Port A or B, each of its bits can be set or reset by using the Bit Set/Reset operation.

d. Interrupt Control Features

When the 8255 is programmed to operate in Mode 1 or Mode 2, control signals are provided that can be used as

interrupt request inputs to the CPU. The interrupt request signals, generated from Port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop. This function allows the programmer to disallow or allow specific I/O devices to interrupt the CPU without effecting any other device in the interrupt structure.

e. Mode 1

This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or 'handshaking' signals. In Mode 1, Port A and Port B use the lines on Port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two transfer ports (A and B)
- Each transfer port contains one 8-bit data port and 4 bits from one half of the control/data port (Port C).
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.

f. Port A

Port A is the most versatile of the three ports. It can be programmed to function in any of the three 8255 operation modes. This first port is the only port in each group that already includes a permanent bidirectional driver/terminator network, 8226 bus driver device at A1 and A2 (group one).

Before Port A is programmed for input or for

output in any of the three operation modes, certain jumper connections must be made to allow the port to function properly in the chosen mode. The 51-52-53 (group 1) jumper pad specifies the direction of data flow for the 8226 bidirectional bus drivers. If output in mode 1 is to be used, jumper pair 52-53 (group 1) should be connected.

When Port A is programmed for Mode 1, interrupts can be used. The INTR output from bit 3 of Port C activates the peripheral I/O interrupt request, PIA1. PIA1 is forwarded to the interrupt logic.

Because the 8226 bus drivers are inverting devices, all data is considered to be negative true with respect to the levels at the J1 or J2 edge connector.

g. Port C

As was described before, the use of Port C depends on the modes programmed for Port A and B. If Port A is in mode 1, bits 3, 4, 5, 6 and 7 of Port C can have the following dedicated control functions.

Bit 3 - INTR (Interrupt Request)

Bits 4 and 5 - Either input or output.

Bit 6 - ACK/ (acknowledge input)

Bit 7 - OBF/ (output buffer full)

2. Parallel Output Priority task

The Parallel output function is performed by a priority task, scheduled by the interrupt handler. The procedure LP\$OUTPUT in PAIO is the priority task, defined as

such by procedure PAIO\$START at system initialization. LP\$OUTPUT dumps the parallel output buffer one character at a time. When the buffer is empty, it checks if the connected module wishes an acknowledge message to confirm the end of the output process, in which case it sends the message.

The process of outputting a character includes: setting the interrupt receiver for the acknowledge, sending the inverted character to the device, and then sending a strobe using bit 4 in Port C.

3. Parallel Output Monitoring Procedures

The Parallel Output priority task is monitored by a set of procedures. These procedures are controlled by the System Message Task, which receives the message requesting service from the task. In the same way as with the serial interface, the priority task is controlled by controlling the buffer, the parallel output buffer in this case. There is no command word controlling the hardware but the first character must be sent by calling procedure LP\$STARTOUT.

4. Peripheral Interface Device

The Intel 8255 Programmable Peripheral Interface Device is programmed for using Port A for output in Mode 1. The proper hardware modifications as described in Reference [5] were done and the Terminator network and driver network installed in sockets A3 and A4 respectively.

G. THE SYSTEM MONITOR

The System Monitor was constructed with the idea of using it as a tool during the debugging of the applications programs. It has been built from a set of commands which will allow the user to examine, substitute, fill or move memory sections in common or on-board memory. It also has commands for changing the allocation of the system printer, for transmitting messages and for changing the allocation of the monitor itself. That is, for passing the connection with the system console from the monitor in one computer to the monitor in another.

There are two ways for setting up communications with the Monitor. One is by pressing the special key, identified by MONITOR\$ID on the system console. The other is using the monitor command GO. If an address is specified following the command G, an interrupt 1 code will be inserted at that address. When the processor gets to this address point (if ever), the interrupt handler will call a procedure that will schedule the monitor to be executed.

All the Monitor procedures are attached to the system message tasks, the same as with the serial I/O and parallel output.

The Monitor commands are:

- D <address 1>,<address 2> : Dumps memory space from address 1 to address 2.
- F <address 1>,<address 2>,<character> : Fills memory locations from address 1 to address 2 with the character

specified.

- M <address 1>,<address 2>,<address 3> : Move memory block bounded by address 1 and address 2 to locations beginning with address 3.

- S <address> : Substitute subsequent memory locations beginning with location pointed by address. The sequence will finish with a carriage return.

- P <number> : Change system printer allocation to computer defined by number. Number must be between 1 and 3.

- T <number 1>,<number 2> : Transmit message <number 2> to message task <number 1>. If transmission succeeds the connection with the system consol is broken.

- C <number> : Change control to the computer indicated by number. Number must be between 1 and 3.

- G [<address>] : Break connection with the system console. Optionally, before leaving, the monitor could put a trap at location <address> for getting back to the system console when the processor gets to that point.

Observe that the Monitor is not active when it is not connected to the system console.

H. SYSTEM'S ADDRESS SPACE

All the program development for both the distributed operating system and the emulator was done using the system's programming language PLM80 under the ISIS-II operating system. The ISIS-II function LINK was used to link the program modules. All the external references that one

module would make were resolved by the LINK function. Function LOCATE, also from ISIS-II, was used to allocate the different program segments into memory. Using this function, all the memory references are resolved and the code is ready to be loaded. The command used was:

```
LOCATE  DISIS.LNK CODE(0000H) MEMORY(C000H) STACK(3040H)
DATA(31D2H) STACKSIZE(100H)
```

The programs were compiled separately for each computer, because of the difference in the system parameter CPCR, that identifies the host computer and the system parameter FIRSTMN, that identifies the number of the system message tasks related to that computer.

The standard SBC 80/20 was changed to increase the on board read only memory capacity. The necessary changes can be found in Reference [5] page 4-3.

I. PROGRAM DESCRIPTION

The Distributed System was divided into fourteen (14) program modules. Each module was placed in a different file. A file may contain one or more public procedures, or may contain public data declarations. Each file was compiled independently. Later, object programs generated by the compiler were linked together. The function of each procedure was described by the comment statement preceding it. All program listings were generated by the PLM/80 compiler.

1. File EXEC

File EXEC contains the program module EXECUTIVE which contains the procedure EXEC and the main program entry.

Procedure EXEC is the Distributed System Executive. This procedure manages the scheduled tasks and is the kernel of the system. As soon as EXEC gains control from the main program body, it calls procedure EXSTART, which initializes the system environment. Then EXEC will continue in a never-ending loop which form the heart of the system. There are three major sections in this loop: the priority task section, the message task section, and the periodic task section.

The priority task section carries out the tasks based on the variable PRIORSCHEDULE. This byte variable works as a vector flag. When a priority tasks needs to be served, the system routine PRIORITY will set a predetermined bit in this variable to one. There are eight priority tasks corresponding to the eight bits in this variable.

When the priority section finds that PRIORSCHEDULE is different from zero, it will look in the address vector PRIORLIST and extract the entry point address of the priority task and execute the task. When the priority task ends the flag bit will be reset to zero.

The message task section works based on the variable NUMMSG and EXTMSG. NUMMSG counts the number of messages in the buffer MSGBUFFER waiting to be processed. The header of

the message contains the destination message task number in the first byte. With this number the EXEC looks into the address vector MSG\$MOD\$ADDRESS for the task entry point and then executes it. In case the received message task is not between the eight assigned values of the executive, the message will be sent to the system buffer using system routine SENEXT. EXTMSG flags the presence of a message for a local message task in the system buffer. When the flag is set to the computer identification number of that single board computer, the executive will move the message to its buffer using system routine RECEXT.

The periodic message section works based on the variable NUMPER. This variable indicates the number of activated periodic tasks. When the number is greater than 0, the executive will examine each activated task and will check if the task activation time has arrived. It does so using system routines COPY\$CLOCK and TIME\$CMP. If the time has arrived, EXEC will pick up the entry point address from the address vector PERLIST and will execute the task. At the end of the task a new activation time will be computed using system routine SETPERTIME.

Priority tasks are checked first and immediately after any other task has been served. In this way message tasks will be served only if there is no priority task scheduled. Periodic tasks will be served only if there is neither a priority task nor a message task scheduled. When there is no task scheduled at all, an external light will be

lighted as an indication that the processor is idle.

The main program function is to check for the initialization variables. LOADSEC is checked to determine the proper moment for internal initialization. START is checked to determine if the computer is required in the actual configuration. Besides this, the main program body saves the initial stackpointer value for a possible reinitialization and sets the computer identification variable CPTB\$ID.

2. File INTMSG

The program module INTMSG includes procedures for the interrupt handlers, the system message procedure, and the computer initialization procedures.

The interrupt handler procedures include a procedure with the attribute INTERRUPT for each one of the six interrupts used by the system, and a procedure INTRESET for the reset of the hardware interrupt mechanism. This module was compiled with the compiler parameter NOINTVECTOR so the loader would not try to construct an interrupt vector in common memory when the module was tested.

Interrupt 0 checks variable CDCACTIVE before calling the required procedure whose entry point address was saved in CDCADR. It also resets counter 1 to mode 0 so it will stop counting and will wait until the next value is loaded.

Interrupt 1 just calls the monitor routine MONITOR\$TRAP that will wake up the local monitor.

Interrupt 2 updates the system clock, which is only updated by computer 1. The variable CLOCK must be checked before modifying the real-time clock in order to ensure that no other processor is in the middle of reading the multibyte clock value. Notice that the four element vector RTC overlays variables RTC0, RTC1, RTC2, RTC3, so that both variables can be used to reference the real-time clock. This was necessary because the code produced for index computation by the compiler would destroy the flag used by the operator PLUS if the vector increment mechanism were chosen.

Interrupt 3, 4, and 5 schedule a priority task by calling the system routine PRIORITY with the proper parameter.

Interrupt 6 restores the stack pointer STACKPTR to its initial value SAVESTACKPTR, resets the initialization variable LOADSBC to 1, and transfers control to SYSSTART.

MSGENT0 is the system message task procedure. It controls the use of the procedures for the serial I/O interface, for the parallel output interface, and for the monitor. Basically it receives messages that require the use of some of the procedures. It checks the validity of the message and then transfers control to the corresponding procedure.

The initialization procedures are controlled by procedure EXSTART. EXSTART calls to SET\$EX\$DATA for data initialization, SET\$EX\$MSG for system messages initialization, and SET\$EX\$INTE for interrupt vector

initialization. Then, it programs counters 0 and 1, and in case computer 1 is in charge, it sets counter 0 with a value that produces the first 1 millisecond interrupt for the real time clock update. Finally, it calls SEIOS\$START for serial I/O initialization, PAIO\$START for parallel output initialization, outputs the initial interrupt mask, and sends the "ready" message to the system console using system routine PRINT\$ASYNC.

3. File SEIO

The program module SE\$IOSMOD contains the procedures for controlling the serial I/O communication. Procedures CSINPUT and CSOUTPUT are the serial input and output priority tasks respectively. The other procedures are called from the system message task MSGENT0 to initialize the hardware and monitor the software over which these procedures perform.

CSINPUT generates inputs from the USART, character by character. It checks for special control characters like: CONSCL\$ID for changing the system console to the local computer by resetting its system messages; MONITOR\$ID for disconnecting from any other task and connecting with the monitor; DELINPUT for deleting the last character received; and CR for detecting the end of the input string and transmitting it to the connected task in case it was requested. If no special character was detected, the input character is saved in the input buffer CRTIN and the output

buffer CRTOUT for later transmission and immediate echoing respectively. CSOUTPUT picks up characters from the output buffer CRTOUT and sends them to the USART. When the buffer is empty, it will acknowledge it to the connected task if required, and then, it will disable the Transmit Enable Signal in the USART status, in order to stop the next scheduling of the task. Procedure STARTOUT is used for triggering the output sequence by enabling the Transmit Enable Signal in the USART.

SEIOSTART initialises the hardware and enters the priority task. The priority tasks are initialized using the system's routine ENTERPRIOR. The returned indices, used by the corresponding interrupts, are saved in variables USART\$IN and USART\$OUT.

All other procedures are used for loading or unloading the I/O buffers in one way or another and for controlling the use of the priority task by only the connected task.

4. File PAIO

The program module PA\$IO\$MCD contains the priority task LP\$OUTPUT, the initialization procedure PAIO\$START, and other procedures for the control of the parallel output buffer LPOUTBUF. LP\$OUTPUT dumps the parallel output buffer character by character. Optionally it can send an acknowledge message to the connected task when the buffer is empty.

PAIC\$START initializes the data and introduces the priority task using system routine ENTERPRIOR. The returning index PRELL\$OUT is later used by the corresponding interrupt handler.

5. File MCNI

The program module MONITOR has the procedures to carry out the monitor commands and the connection with the user tasks.

Procedure MONI\$INTER is the command input procedure. It gets the command data string from the data portion of the message sent by the user task to MSGENT0. It uses system routine GET\$CHAR to extract the command's first letter. Upon comparison with the possible command codes it decides whether to transfer control to the corresponding procedure or ignore the command because it is invalid. After a command has been carry out or refused, the procedure will send a message input request for the next command using procedure RECEIVE\$NEXT\$COMMAND. This procedure uses system procedures PRINT\$SYNC to output the prompt character '.', and the system procedure CON\$INPUT\$REQ for requesting an input message from the system console.

6. File SC1

Program module SCPUP1 has the system routines of the lowest level. They are directed to set and check working data. The system routines are: FILL, CLEARDATA, SHFT, BIT,

CLEARBIT, SETBIT, and LEGAL.

- FILL(address1,address2,character). This routine will fill out memory from location <address1> to location <address2> with the character <character>.

- CLEARDATA(address1,address2). This routine will fill out memory from location <address1> to location <address2> with a blank character.

- SHFT(bit-position). This routine will return an 8-bit word with a 1 in position <bit-position>.

- BIT(bit-position,byte). this routine will return TRUE if the bit <bit-position> in byte <byte> is set. FALSE otherwise.

- CLEARBIT(bit-position,byte). This routine sets bit <bit-position> in byte <byte> to zero.

- SETBIT(bit-position,byte). This routine sets bit <bit-position> in byte <byte> to one.

- LEGAL(byte1,byte2,code). This routine returns TRUE if the value in byte <byte1> is lower than the value in byte <byte2>. Otherwise, it returns FALSE and outputs an error with code <code>.

7. File SC2

The program module SCPUB2 contains the system routines related to the periodic tasks and the system clock. It includes the system routines: COPY\$CLOCK, SETPERTIME, TIMESCMP, PERACT, PERCHG, and PERSUSP.

- COPY\$CLOCK(base). This routine copies the real

time clock into a four byte vector based in <base>.

- SETPERTIME(index). This routine sets the next activation time for the periodic task indicated by <index>. To do this, it will add the actual time, copied from the real time clock, to the period of the task and then it will save this result into the task activation time vector PERLIST(<index>).PERTIME.

- TIMES\$CMP(base1,base2). This routine will return a TRUE if the four-byte vector based on <base1> is lower than the four-byte vector based in <base2>. FALSE otherwise.

- PERACT(address1,address2). This routine will define a periodic task by filling out information on the periodic task data structure PERLIST. PERLIST(index).PERADR will contain the procedure address given by <address1>, PERLIST(index).PERINTADR will contain the address of the four-byte vector with the period given by <address2>.

- PERCEG(index,address). This routine is used to change the period of the periodic task indicated by <index> to the period indicated at the vector at location <address>.

- PERSUSP(index). This routine is used to suspend a periodic task. To do so, it is sufficient to define the data element PERLIST(<index>).FREE as TRUE and rearrange the index in the list PERXTBL.

8. File SC3

The program module SCPUB3 contains the system routines related to the message tasks. It includes the

system routines: MSGOVERFLOW, PACKMCB, SEND, GET, RELEASE, SETEXTMSG, RECEXT, SENEXT, and ILLEGALMSG.

- MSGOVERFLOW. This routine outputs the error code 01 for message buffer overflow.

- PACKMCB(address). This routine packs a four-byte vector from location <address> into the local message buffer MSGBUFFER.

- SEND(address). This procedure sends a message to a message task. It checks if the message task of the receiver as well as the sender is active, puts the header, located at <address> into the buffer, sets the pointer ADRMSGDATA to the location following to the header in the message buffer and returns a TRUE if this sequence has succeeded. Otherwise, it returns a FALSE.

- GET(semaphore). This routine will get the <semaphore> for the computer.

- RELEASE(semaphore). This routine release the <semaphore> from the computer.

- SETEXTMSG. This routine sets the value of EXTMSG for the use of the system buffer EXTMSGBUFFER.

- RECEXT. This routine moves a message from the system buffer EXTMSGBUFFER to the local buffer MSGBUFFER.

- SENEXT. This routine moves a message from the local buffer MSGBUFFER into the system buffer EXTMSGBUFFER.

- ILLEGALMSG. This routine displays the illegal message signal on the system console.

9. File SC4

The program module SCPUB4 contains the system routines related to the priority tasks and interrupts. It includes system routines: PRIORITY, ENTERPRIOR, REMPRIOR, SETCDC, ENTER\$INT, and REM\$INT.

- PRIORITY(index). This routine is used to schedule the priority task indicated by <index>.

- ENTERPRIOR(address). This routine is used to enter the procedure whose entry point is at <address> as a priority task. The routine will return an index corresponding to the priority task.

- REMPRIOR(index). This procedure is used to suspend the priority task indicated by <index>.

- SETCDC(2-word-value,address). This routine is used to define a procedure at <address> that will be executed after .930 x <two-word-value> micro-seconds.

- ENTER\$INT(index). This routine is used to enable the interrupt indicated by <index>.

- REM\$INT(index). This routine is used to disable the interrupt indicated by <index>.

10. File SC5

The program module SCPUB5 contains the system routines related to the system console and the system printer. It includes the system routines: PRINT\$ASYNC, PRINT\$SYNC, CON\$INPUT\$REQ, CON\$LINK\$REQ, PAR\$LINK\$REQ, CON\$RELEASE, CON\$NEW\$LINE, CON\$BEG\$LINE, PRINT\$LINKED,

PAR\$RELEASE, PAGE, WRITE, and WRITE\$LINKED.

PRINT\$ASYNC(address,count). This routine sends to the system console <count> bytes from text at location <address>.

- PRINT\$SYNC(address,count,index). This routine sends to the system console <count> bytes from text at location <address> and the sender <index> that must correspond to the connected task.

- CON\$INPUT\$REQ(index,flag). This routine request an input string message from the system console. The message should be sended to message task <index> and the screen rotated or not depending on <flag>. The receiver must be 'connected'.

- CON\$LINK\$REQ(index). This routine connects message task <index> with the system console.

- PAR\$LINK\$REQ(index). This routine connects message task <index> with the system printer.

- CON\$RELEASE. This routine disconnects the system console.

- CON\$NEWS\$LINE(index). This routine produces a new line on the system console. The caller must identify himself by <index>.

- CON\$PEG\$LINE(index). This routine produces a new page on the system console. The caller must identify himself by <index>.

- PRINT\$LINKED(address, count, index, code). This routine displays <count> bytes from location <address> into

the system console. <index> must identify the connected task. The system console will send back a message with the data <code> in it.

- PAR\$RELEASE. This routine disconnects the system printer.

- PAGE(index). This routine produces a new page on the system printer. The caller must identify himself by <index>.

- WRITE(address,count,index). This routine displays <count> bytes from location <address> on the system printer. <index> must identify the connected task.

- WRITE\$LINKED(address, count, index, code). Same as WRITE, but the system printer will send back a message with the data <code> in it.

11. File SC6

The program module SCPUB6 contains system routines for I/O computations mainly. It includes system routines: ACTIVE, UPDSTAT, CONVASC, DEBLK, GETNUM, VECTOR\$ADD, VECTOR\$SUB, NUMOUT, and GETCHAR.

- ACTIVE(index). This routine will return TRUE if message task number <index> is active.

- UPDSTAT(index,status). This routine will update the status of message task number <index> to <status>.

- CONVASC(byte). This routine will convert a 2-digit hexadecimal number <byte> into 2 ASCII characters. The characters will be left in the 2-word variable A4, least

significant bit first.

- DEBLK. This routine will skip the leading blanks in an input string message.

- GETNUM. This routine transforms an ASCII string into an hexadecimal value. The hexadecimal value will be stored into variable A4. The routine reference is used as a flag: it returns a TRUE if the transformation was possible, FALSE otherwise. The ASCII string is taken from the message data and it must not exceed the decimal value of sixty five thousand five hundred thirty six (65536).

- VECTOR\$ADD(address1, address2, address3). This routine adds the four-byte vector at <address1> to the four-byte vector at <address2> and puts the result at the four-byte vector at <address3>.

- VECTOR\$SUB(address1, address2, address3). This procedure subtracts the four-byte vector at <address2> from the four-byte vector at <address1> and puts the result into the four-byte vector at <address3>.

- NUMOUT(value,base,lc,address,width). This routine converts the value <value>, currently in base <base>, into an ascii string of at most <width> characters, and stores the string at location <address>. The unused left spaces are filled with the character <lc>.

- GET\$CHAR. This routine gets an ASCII character from the input string message.

12. File SC7

The program module SCPUB7 contains three procedures for three system routines: ENTER\$MSG\$MOD that defines a new message task; MONITOR, that request the connection with the monitor; and ERROR that outputs an error message on the system console.

13. File MODPRO

The File MODPRO contains the procedures ENTER\$MOD and CLEAR\$MOD that are offered to the user for message definition under ISIS-II operating system. This file is supposed to be included in the user programs by means of the INCLUDE directive of the ISIS-II operating system.

14. File RUN

This file contains the program RUN that sets the initialization parameters LOADSBC and START and puts the MDS CPU into HALT state. RUN uses ISIS-II routines for reading the number of the computers the user wants to trigger from the console. It also constructs an interrupt handler environment that only allows interrupt six to be served from the MDS. To return to the ISIS-II, the MDS must be rebooted. The interrupt handler resets the interrupt request to allow the user to use interrupt 6 again.

IV. TACTICAL SYSTEM EMULATOR

A. DESIGN

The Tactical System Emulator is based on the idea that a Tactical System can be partitioned into functional modules and that each of these functional modules may be replaced by dummy modules that will demand the same processing time and intermodule communication effort from the hardware.

Replacing the real tactical system by a dummy system will allow us to study the behaviour of the hardware, in this case a distributed micro-computer architecture, without programming each module. The system designer will only concern himself with the definition of the functional parts that constitute his tactical system and then exercise the emulation in order to study the proposed system's behaviour.

Two kinds of functional modules are recognized: Periodic functional modules and Demand Functional Modules. Periodic Functional Modules are activated periodically by the controlling mechanism and are usually in charge of updating data structures or pooling data from or to some peripheral. Demand functional modules are activated upon demand from another functional module, these are in charge of most of the computations.

1. Dummy Function

All functional modules will be replaced by dummy tasks (periodic or message) and data structures which

determine their performances. These tasks will be directed to consume an amount of processor time similar to the the estimated execution by the functional module they are replacing. Also they transmit the same messages, with the same destination and length as the modules which they replace.

The data block that contains the information on the basis of which the dummy task emulates the functional module is conceptualized as an activation record. All functional modules that are defined in the system will have one activation record. The dummy tasks do not need to be all different because they will perform based on different activation records. In fact, only two (2) dummy tasks are needed: one for the periodic functional modules and one for the demand functional modules.

All activation records include:

- Functional module identification number
- Processor time needed
- Number of message that are transmitted
- Destination, Length and number of each message
- Total Time in Execution
- Number of times Activated

Besides this, periodic activation records contain:

- Period (in real-time clock format)

The demand activation record contains:

- Number of States Before Execution
- Number of input messages

- Message number, number of times received and total delay for each input message.

2. Emulation Sequence

There are three very definite steps in using the emulation system:

The first one, as already mentioned, is identifying the functional components of the tactical system.

The second one, is creating the dummy system by interacting with the dynamic tactical system definition tool. In this interaction the user defines the real system parameters to the emulator.

The third step is running and collecting statistics from the emulator. After the statistics have been collected, the user can go back to step 2 and modify the parameter or move the allocated dummy functions from one computer to another.

B. IMPLEMENTATION

Each computer will be able to handle up to eight periodic and eight demand dummy modules. The corresponding activation records were allocated in on board memory and the code in common memory. In this way all three computer will share the same code for the dummy modules and the activation record over which they perform will be different for each processor. The same is true with the code that dumps the statistics from the activation records.

1. Emulator-System Relation

The emulator uses message task three (3) for controlling the emulation operation. This message task will connect itself with the system console from which it will get the input message coming from the user. Message 3 will receive the real tactical system parameters together with the emulator environment the user wishes to use. Message 3 will send these parameters to message tasks in the other computers in order to make them construct the activation records.

Message task number 2, 10, and 18 in computers 1, 2, and 3 respectively were used to construct the activation record. They receive the necessary information from message task 3. Message tasks 1, 9, and 17 are the dummy demand modules for computers 1, 2, and 3 respectively. Observe that the three message tasks use the same code in common memory. The same is true for message tasks 2, 10, and 18.

Message tasks 4, 12, and 20 were used for dumping the data and statistics in the activation record. They become active upon receipt of a message from message task 3 after the user has requested the dump. They are also used to move the activation record from one computer memory to another, corresponding to moving a dummy module from one processor to another.

Message tasks 3, 4, 12, and 20 were defined in the main body of the emulator program. They will be executed under ISIS-II operating system so that the procedures used

for the definitions were from program MODPRO. Message tasks 1, 9, 17, 2, 10, and 18 were defined in procedure SET\$MODS, executed upon receipt of the system initialization message by message number three. Procedure SET\$MODS uses system routine ENTER\$MSG\$MOD for this purpose.

2. Emulation Control

The emulation is initiated from the system point of view by activating the periodic tasks that replace the periodic functional modules. They are supposed to trigger or initiate the message sequences that would activate the system.

Periodic tasks are activated by message task 2, 10, and 18 upon receipt of the commanding message from message task 3. They use system routine PERACT(Address,Period) for this purpose.

The emulation can be stopped in two ways: First, the user could define a 'sink' module that would count the number of times the same message has been sent to him, and when this count gets to a predetermined number it would send a message to the system that would suspend all periodic tasks and cancel all message task. Second, the user could specify a time limit, using the count down clock CDC. The emulator would count to this limit and when reached, it would produce the same stopping message as in the previous case.

3. Linking and Locating

The emulator programs must be linked together as well as the referenced system routines. Because system routines were already located in local Read Only Memory, the linking must be done with the PUBLIC references of the distributed system code. The linking command for the emulator was:

```
LINK :F1:EMULA1.OBJ, :F1:EMULA2.OBJ, :F1:EMULA3.OBJ,  
:F1:EMULA4.OBJ,      SYSTEM.LIB(EXIT),      SYSTEM.LIB(ISIS),  
PUBLICS(DISI), PLMS0.LIB TO :F1:EMULA.LNK
```

The emulator code was located in common memory starting at 5000H to allow the code to be shared by more than one processor. The emulator data must be located between locations 3900H and 4000H. The command used was:

```
LOCATE :F1:EMULA.LNK CODE(5000H) DATA(3900H)
```

C. PROGRAM DESCRIPTION

1. File EMULA1

This program module contains PERIODIC\$MOD and DEMAND\$MOD, the periodic and demand dummy procedures, the ones that will be executed instead of any periodic or demand functional module that exist in the real tactical system. It also has DATA\$FILLER, the procedure that is shared by message tasks 2, 10, and 13.

The procedure SEND\$MSG(S(Base)) is used to send the messages between the dummy functions. It takes the

information from the activation records and only requires a base to the variable NUM\$MSG\$OUT.

The procedure EMULATE(Base) consume an amount of times dictated by the variable pointed to by <base> and increments the execution time and total executed time in the corresponding variable.

CORRECT\$INPUT\$MSG verifies that the message received is correct and updates the statistics about that message. It receives the coming message identification number and a base to the coming message list.

CORRECT\$ID checks whether the scheduled function to be emulated is correct or not and sets the activation record pointer MODE\$BLOCK.

Observe that in this program, the mnemonics DBD, DBP, DTA, and MPK have been used instead of the references DATA\$BLOCK\$DEM, DATA\$BLOCK\$PER, MSG\$DATA, and MOD\$BLOCK respectively. Procedures SEND\$MSG, EMULATE, and the CORRECT\$INPUT\$MSG are used by both the DEMAND\$MOD and PERIODIC\$MOD.

2. File EMULA2

This module contains the emulator program's main program, the emulator coordinator task, and procedures for the control of the principal functions of the emulator.

The main program body will be executed under ISIS-II operating system. It uses procedures CLEAR\$MOD and ENTER\$MOD to initialize the emulator message tasks and the ISIS-II

system routine EXIT to give the control back to the user after this initialization. Observe that the file :F1:MODPRO.SRC has been included as part of the module and that the routine EXIT has been declare EXTERNAL.

MSGENT3 serves as a coordinator for the emulator to establish the relation with the user through the system console and the line printer and it directs the steps of the emulation. Upon receipt of the initialization message (message number 0), it will call INIT\$EMU. INIT\$EMU initializes the other emulator message tasks by calling procedure SET\$MODS, clears its local data using the system routine CLEAR\$DATA, and request the use of the console by using the system routine CON\$LINK\$REQ. The system responds to the console with the message number 21. When MSGENT3 receives message 21 it calls procedure CHECK\$SE\$CONNECTION. This procedure will keep asking from the system console if the previous request was denied or it will initiate the interaction with the user for setting up the parameters for the emulation. CKECK\$SE\$CONNECTION asks the first response from the console by activating the routine CON\$INPUT\$REQ. The inputs from the console will came with message number 20. When this message number is received, MSGENT3 will call RECEIVE\$INPUT, that belongs to programming module EMULA4.

When MSGENT3 receives the message number 30 it will stop the emulation by calling the procedure STOP\$EMU. This procedure sends message to message task 2, 10, and 18 that will suspend the periodic tasks and it also will deactivate

the message task used as dummy functions. Besides this, it will also read the finishing time and compute the time taken by the emulator and put it into EMU\$CLOCK\$2.

EMULA2 also contains procedures for the control of the principal functions of the emulator, they include: Start Emulation (START\$EMU), Move task (MOVE\$TASK), Substitute data (SUBS\$DATA), and Write data and statistics (WRITES\$DATA).

START\$EMU sends the triggering message for the activation of the periodic tasks (it initiates the emulation), and saves the emulation initialization time in variable EMU\$CLOCK\$1. In case a time limit emulation is chosen, START\$EMU defines a priority task that will be called after the "count down clock" event. Procedure EMU\$COUNTER will be a priority task and it will be scheduled every fifty milliseconds.

MOVE\$TASK moves an activation record to another computer. It receives its data from the message data, the first byte of which should be the activation record number that must be moved, and the second byte the receiving computer number. With these two bytes, and examining the data structures ID\$TBL and MSG\$TBL, MOVE\$TASK constructs the appropriate message with the information for constructing the activation record in the receiving computer. The message is sent to message tasks 4, 12, or 20 (procedure MSGENT4).

SUBS\$DATA substitutes previously sent data. It interprets the command and directs the user to the iteration

point at which the data was sent, so that the process is repeated.

WRITE\$DATA interprets the writing command and then uses the procedure SEND\$WRITE\$MSGs to send the appropriate message to MSGENT4.

3. File FMULA3

The programming module EMULA3 includes the procedures for writing out the real system parameters and the collected statistics on the system console or the line printer. It also includes the procedure for reordering the activation record in local memory. Both functions are controlled by procedure MSGENT4, used by message tasks 4, 12, and 20.

MSGENT4 will set up a print flag vector variable called PRN\$FLAG and then will call procedure SET\$WRITER which in turn will establish connection with the system console or the line printer; this is based on the message number and the data byte received with it. When the connection is acknowledged with message number 28, procedure PRN will print out the desired data. PRN prints the data, sending messages in batches. The last message of a batch requests an acknowledge from the printing message task. The acknowledge message is message number 26. When MSGENT4 receives message number 26 it just calls procedure REC\$COSTBCODE which continues with the next batch.

MSGENT4 also receives the message that trigger

procedure SEND\$TASK that sends a required activation record to a specified computer, REC\$TASK that receives the activation record coming from another computer, and SET\$NEW\$SINK that changes the destination of all the messages directed to the dummy function that uses the moved activation record.

4. File EMULA4

The program module EMULA4 contains the procedures needed for the interaction with the user in order to receive the real time system and emulation parameters. All these procedures are directed by procedure RECEIVE\$INPUT that is called by MSGENT3.

The program EMULA4 has several entry points. The entry point a particular activation gets to, depends on the state of the interaction with the user. This state is controlled by saving a value in variable STATE.

When variable STATE has the value 0, the entry point will be procedure STATE0; when variable STATE has the value 1, the entry point will be procedure STATE1; and so on. There are up to 32 possible values for variable STATE with its corresponding entry point procedures. The last entry point, corresponding to a value equal to 32 in variable STATE, is procedure STATE32.

The variable STATE is modified after a particular question has been responded by the user, or after a limit has been reached.

V. CONCLUSIONS

Multi-micros can be used by dividing a tactical application into explicit asynchronous (but cooperating) processes. The emulator makes it possible to evaluate alternative process assignments (to processors) as an aid in designing the process structure. Future work is required to evaluate how many processors can be effectively used.

There is a need for an application independent operating system for multi-micros. Much of this thesis effort was dedicated to tailor the special purpose operating system on hand. The operating system should manage global/local memory. A more efficient emulator could be constructed if the operating system provides a mechanism for translating the dummy functions code between local and global memory.

There is a trade-off between using more memory for storing similar copies of a code in each computer local memory or increase the possibility of bus contention by sharing code in common memory.

The emulator design is not tied to whether or not the functional modules are in local or global memory, but this implementation links and locates programs in a way that the emulator only represents structures of multiple computers sharing code in common memory.

An emulator for structures which use common memory strictly for interprocess communication can be derived from the same programs by locating the module EMULA1 in on-board

memory. In order to locate EMULIA1 in on-board memory, some modules from the O/S must be disregarded, because of the limited space. Modules PAIO and MONI are the most likely to be disregarded for this purpose.

APPENDIX A. SBC 80/20-4 DESCRIPTION

The SBC 80/20-4 is a member of Intel's complete line of OEM computer systems which take full advantage of Intel's LSI technology to provide economical, self contained computer based solution to OEM applications. The SBC is a complete computer system on a single 6.75-by-12 inch printed circuit card. The CPU, system clock, read/write memory, non-volatile read-only memory, I/O ports and drivers, serial communication interface, interval timer, interrupt controller, bus control logic and drivers all reside on the board.

To facilitate the following description, the SBC 80/20 can be divided into eight major functional blocks:

- 1) CPU Set
- 2) Bus Interface
- 3) Random Access Memory
- 4) Read Only Memory
- 5) Serial I/O interface
- 6) Parallel I/O interface
- 7) Interval Timer
- 8) Interrupt Controller

The CPU Set consist of the 8080A Control Processor, the 8224 Clock Generator and the 8238 System Controller. The CPU Set is the heart of the SBC 80/20. It performs all system processing functions and provides a stable timing reference for all other circuitry in the system. The CPU generates all

of the address and control signals necessary to access memory and I/O ports both on the SBC 80/20 and external to the SBC 80/20. The CPU set is capable of fetching and executing any of the 8080's instructions. The CPU set responds to interrupt requests originating both on and off the SBC 80/20, and to WAIT requests from memory or I/O devices having an access time which is slower than that required by the SBC 80/20's 8080A cycle time.

The Bus Interface allows the SBC 80/20 to use a common system bus with other master devices such as other CPUs or DMA devices, thus sharing common memory and I/O resources. The Bus Interface includes an Intel Bus Controller, as well as circuits for the generation of the Bus Clock signal (BLCK/). The Bus Controller arbitrates all SBC 80/20 request for use of the system bus, synchronously with respect to the Bus Clock. When the SBC 80/20 acquires control of the Bus, The Bus Controller generates the appropriate memory or I/O command signal, gates the address into the system address lines and gates data on/off the system bus. Operation between the CPU and on board resources require no use of the system bus.

The Random Access Memory (RAM) section provides the user with 4096 (2K) x 8-bits of read/write storage, using eight Intel 2114 static RAM devices. The 2114 requires neither refreshing nor clock input to operate. All operations between the CPU and on board RAM require no WAIT states.

The Read Only Memory (ROM) section provides the user

with the necessary provisions for installing 4096 x 8-bit or 8192 x 8-bits of ROM or EPROM. Each SBC 80/20 has four 24-pin sockets that can accept either Intel 8708 Erasable and Electrically Programmable Read Only Memory (EPROM) chips (1024x8-bits each) or Intel 8308 static MOS Read Only Memory chips. Each SBC 80/20 also includes the option of installing four 2048 x 8-bits Intel 2716 Erasable and Electrically Programmable Read Only Memory Chips. The board includes the necessary acknowledge and address decoding circuitry. All CPU accesses to this on-board ROM/EPROM area require no CPU WAIT states.

The serial I/O interface, using Intel's 8251 USART device, provides a full duplex RS232 serial data communication channel that can be programmed to operate with most of the current serial data transmission protocols. Synchronous or asynchronous mode, baud rate, character length, number of stop bits and the choice of even, odd or no parity are all program selectable.

The Parallel I/O Interface, using the Intel's 8255 Programmable Peripheral Interface device, Provides 48 signal lines for the transfer and control of data to or from peripheral devices. Sixteen lines already have a bi-directional driver and termination network permanently installed. This bi-directional network allows these sixteen lines to be inputs, outputs, or bi-directional (selected via jumpers). The remaining 32 lines, however, are uncommitted. Sockets are provided for the installation of driver or

terminator networks as required to meet the specific needs of the user system.

The Interval Timer capability is implemented with an Intel 8253 Programmable Interval Timer. The 8253 includes three 16-bit BCD or binary counters which can be programmed by the user to perform a variety of timing functions. The output from the first two counters can be used as interrupt request lines, thus allowing simple implementation of such features as a real time clock or a program monitor alarm. The output from the third counter is applied to the Serial I/O interface. When properly programmed, this counter can provide the desired baud rate frequency for serial communications.

The SBC 80/20 also includes an Intel 8259 Interrupt Controller. The 8259 device resolves priority among eight different interrupt levels according to an algorithm that is programmed by the user. A jumper area in the interrupt section permits the user to connect any of nine external bus interrupt lines or 17 on-board interrupt request to eight priority levels inputs to the 8259. Thus, by jumpering various interrupt lines to the appropriate priority level and by programming the 8259 with the desired algorithm, the user can easily configure a custom interrupt structure.

APPENDIX B. MULTIBUS DESCRIPTION

A significant measure of the INTELLEC MDC System's power and flexibility can be attributed to the design of its bus. The bus structure allows for multiple master-slave relationships between the various system modules. In fact, the bus can support eight masters in a parallel, priority network. By connecting adjacent masters modules with a serial bus priority line, the maximum number of masters can be expanded to 16. In such a configuration, each master pair contends for bus control, the two masters in the pair further resolve contention via the serial priority line. This configuration allows for an increased number of master modules without incurring the timing overhead of a pure serial network. Where a pure serial bus control network is implemented on the INTELLECT MDS Bus, the maximum bus transfer rate of 5 MHz can't be guaranteed in that application.

The Bus provides its own clock which is derived independently from the processor clock. The Bus clock provides a timing reference for resolving bus contention among multiple bus requests. This feature allows different speed processors to share resources on the same bus. Actual transfer via the bus, however, proceeds asynchronously with respect to the bus clock. Thus, the transfer speed is dependent on the transmitting and receiving devices only. This design prevents slow master modules from being

handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. Once a bus request is granted, single or multiple read/write transfers can proceed at a maximum rate of 5 MHz. The most obvious application for the master-slave capabilities of the bus is multi-processor configurations and high speed direct-memory-access (DMA) operations, but are by no means limited to these two.

The INTELLEC MDS System Bus (excluding power inputs) consist of 56 signal lines, including 16 address lines, 16 bidirectional data lines, and 8 multi-level interrupt lines. Thus, the system is capable of supporting 64K (65,536) words of storage.

APPENDIX C. HOW TO USE THE EMULATOR

The Tactical System Emulator is a tool designed to help the user in the allocation of tactical system functional modules into a distributed micro-computer architecture.

A Tactical System is viewed as a set of functional modules that interact with each other seeking the same goal. Two functional module types are recognized: Periodic and Demand. Periodics will be activated by the system in a constant period and Demands will be activated upon receive of a message with data input.

The emulation sequence consist of three steps:

- (a) the definition of a network that characterizes the process structure of the tactical system,
- (b) the definition of the network parameters to the emulation system,
- (c) the control of the emulation through emulation commands.

A. NETWORK DEFINITION

Every Tactical System can be partitioned into functional modules that are executed as separate processes. This package does not support the software for helping in the definition of a tactical system network, but several methods can be found in References [3] and [4].

The emulator supports up to eight periodic modules and eight demand modules in each computer. Each module can have

up to four sending message numbers, each message number with a single destination. Each demand module can have up to four receiving message numbers.

B. PARAMETER DEFINITION

The emulation parameters are requested and checked by the emulator system itself. Upon system initialization the emulator will be switched automatically to the input state where the user will specify the number of computers needed and the modules allocated in each computer with its corresponding parameters.

The initialization sequence is as follows:

- Bootstrap ISIS-II operating system in MDS Micro-computer Development System. The hardware should include the standard boards with 64K of memory and the required number of Single Board Computers connected. Each SBC with the corresponding copy of the Distributed System on ROM.

- Run the program EMULA. (EMULA contains the emulator code)

- Run the program RUN with the number of the SBCs you want to use. (e.i. "RUN 1 2" will authorize SBC 1 and SBC 2).

- The MDS console will be in HALT, and a the emulator identification tag will appear in the Distributed System Console.

- From there on, you must respond to the emulator

requests in order to define the network.

C. EMULATION CONTROL

After the network parameters have been defined the emulator will put itself into a wait state and will display the possible commands it is able to execute. They include:

W <H><P/D/S><1/2/3>	- write data
E	- emulate
S <#/1/2/3/P>	- substitute data
M #<,1/,2/,3>	- move task number #

"W" is a request for displaying data. If only "W" is keyed, all the periodic module, demand module, and statistic data from computers 1, 2, and 3 will be displayed in the system console. If letter "H" is specified the output will be directed to the line printer. P, D, or S, specify a subset to the data corresponding to periodic, demand or statistics data. 1, 2, or 3 specifies a subset of the data corresponding to computers 1, 2, or 3.

"E" initiates the emulation. At the end of the emulation, the same commands will be available.

"S" is used to change some of the network parameters. If it is keyed alone, the input process will be repeated from the beginning. Otherwise, a specified module number #, computer 1, 2, or 3, or the emulation parameter (Time limit or not) data will be changed.

'M' moves module number # to computer 1, 2, or 3.

DISTRIBUTED SYSTEM PROGRAM LISTING

The following table summarizes the content of the distributed system files.

No.	File name	Procedure	Page
1	run	run	96
2	exec	executive	98
		exec	111
		sysstart	115
3	intmsg	intmsg	116
		intreset	119
		int0	119
		int1	119
		int2	120
		int3	120
		int4	121
		int5	121
		int6	121
		msgent0	123
		setsex\$data	124
		setsex\$msgs	125
		setsex\$inte	126
		ex\$start	127
4	seio	se\$iosmod	129
		csconvxy	133
		packertout	134
		pack	134
		csoutput	135
		startout	136
		termio	136
		csinput	137
		seprint\$sync	140
		not\$act	141
		se\$input\$req	141
		se\$print\$sync	141
		se\$print\$linked	143
		se\$inp\$activate	144
		pos\$cur	145
		se\$newsline	145
		se\$beg\$line	146
		init\$usart	146
		se\$iosstart	147
5	paio	pa\$iosmod	149
		lp\$output	151
		lp\$startout	251
		lppack	153
		lp\$newsline	154

		pa\$newpage	154
		pa\$print	155
		pa\$print\$linked	156
		pa\$out\$active	157
		pa\$out\$release	157
		paio\$start	158
6	moni	monitor	159
		receive\$next\$command	160
		get\$parameters	160
		print\$line	161
		change\$computer	261
		filler	261
		display	163
		move\$memory	163
		send\$mem	164
		sub\$next	164
		substitute	165
		going	165
		transmit	166
		change\$printer	166
		reset\$instru	167
		monitor\$trap	167
		moni\$inter	168
		moni\$init	169
		moni\$ender	169
7	sc1	scpub1	170
		fill	170
		cleardata	171
		shift	172
		bit	173
		clearbit	174
		setbit	175
		legal	176
8	sc2	scpub2	177
		copy\$clock	187
		set\$ptime	179
		time\$cmp	180
		peract	181
		perchg	182
		persusp	183
9	sc3	scpub3	185
		msgoverflow	186
		packmcb	187
		send	188
		get	199
		release	191
		setextmsg	192
		recext	193
		sendext	194
		illegalmsg	196

question has been responded by the user, or after a limit
has been reached.

91

10	sc4	scpub4	198
		priority	199
		enterprior	200
		remprior	201
		setcdc	202
		enter\$int	203
		rem\$int	204
11	sc5	scpub5	205
		print\$async	205
		print\$sync	206
		con\$input\$req	206
		con\$link\$req	208
		par\$link\$req	208
		con\$release	209
		con\$new\$line	209
		con\$beg\$line	210
		print\$link	211
		par\$release	212
		page	212
		write	213
		write\$link	214
12	sc6	scpub6	215
		active	215
		updstat	216
		convasc	217
		deblk	218
		getnum	219
		vector\$add	221
		vector\$sub	222
		numout	223
		get\$char	224
13	sc7	scpub7	225
		enter\$msg\$mod	226
		monitor	226
		error	227

AD-A072 578

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/6 9/2

A TACTICAL SYSTEM EMULATOR FOR A DISTRIBUTED MICRO-COMPUTER ARC--ETC(U)

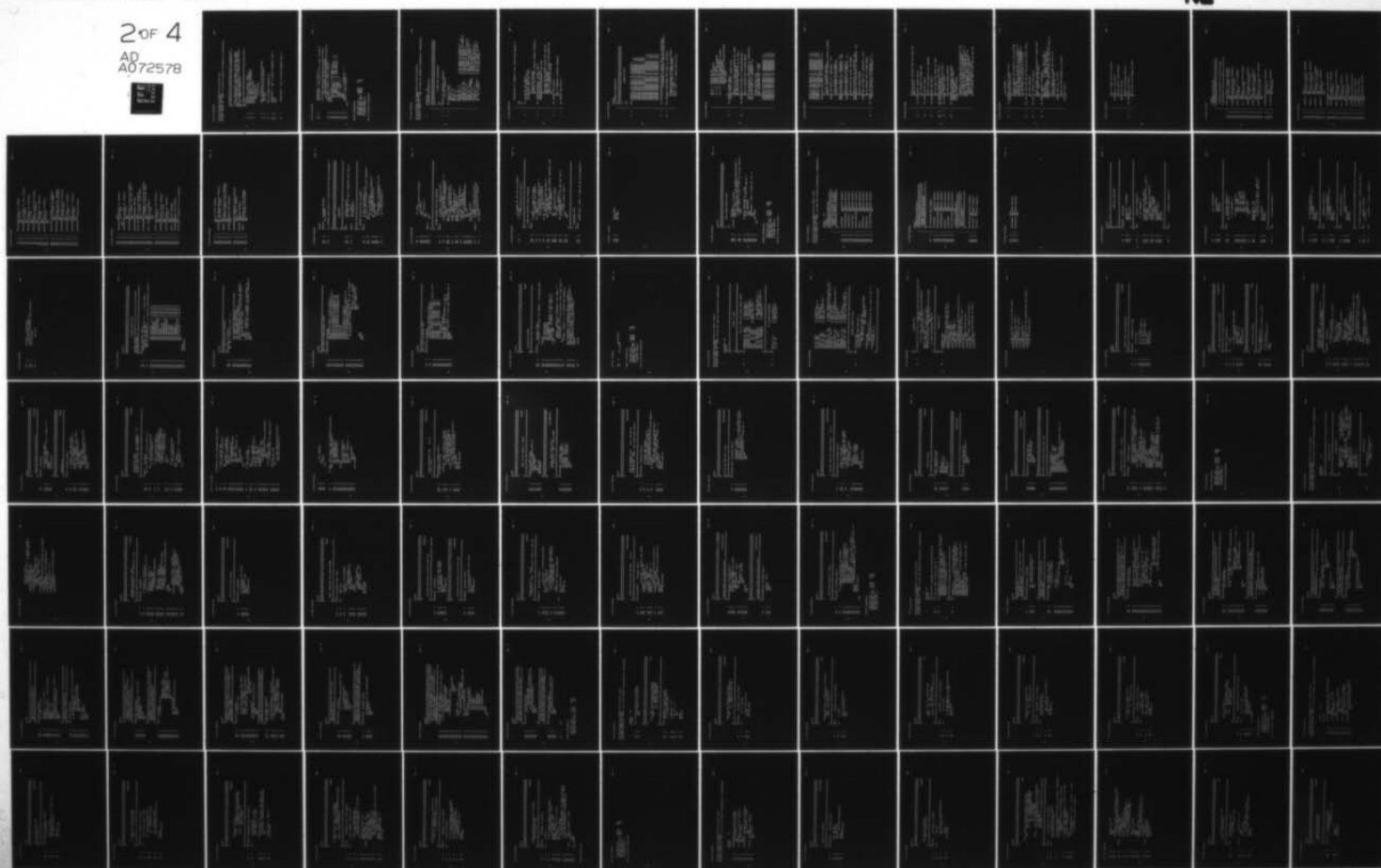
JUN 79 L A GUILLEN

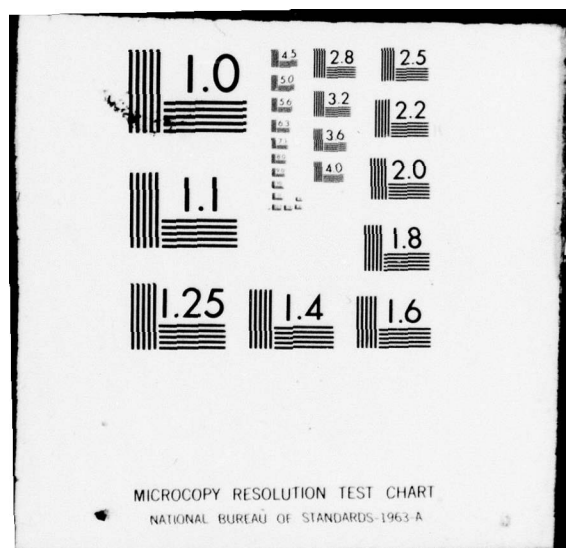
UNCLASSIFIED

NL

2 of 4

AD
A072578





96

```

15 1 OUTPUT(0FCH) = 0 ; /* DITTO */
16 1 OUTPUT(0FCH) = 10111101B; /* INT. 6 AND 1 ALLOWED */
    /******
    READ PARAMETERS AND SET VARIABLES
    ******/
17 1 CALL READ(1,.INBUF,5,.ACTUAL,.STATUS);
18 1 COUNT = 0 ;
19 1 DO WHILE ACTUAL <> 0 AND COUNT < 5 ;
20 2 IF INBUF(COUNT) = '1' THEN STAKT(0) = 1 ;
22 2 IF INBUF(COUNT) = '2' THEN STAKT(1) = 2 ;
24 2 IF INBUF(COUNT) = '3' THEN STAKT(2) = 3 ;
26 2 ACTUAL = ACTUAL - 1 ;
27 2 COUNT = COUNT + 1 ;
28 2 END;
29 1 LOADSBC = 1; /* TRIGGER SBC 1 */
30 1 HALT; /* HALT COMPUTER */
31 1 END;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 009DH      157D
VARIABLE AREA SIZE = 0008H      8D
MAXIMUM STACK SIZE = 0008H      8D
48 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EXECUTIVE
 NO OBJECT MODULE REQUESTED
 COMPILER INVOKED BY: PLM80 :F1:EXEC.SRC NOOBJECT PAGELLENGTH(39) PAGEWIDTH(90)

```

1      EXECUTIVE: DO;
      $ INCLUDE(:F1:SYDATP.SRC)
/*
*****
*****
*****
*****
*****
*****
*****
**
***
***
**/
      DECLARE LIT LITERALLY 'LITERALLY',
              DCL LIT 'DECLARE';

2      1

3      1      DCL TRUE LIT '0FFH',
              FALSE LIT '0',
              RMN LIT '0',
              SMN LIT '1',
              MN LIT '2',
              ML LIT '3',
              MAXCPTR LIT '3',
              MAXMOD LIT '8',
              MAXSYSMOD LIT '24',
              MSGBUFLN LIT '512',
              MAXPRIOR LIT '8',
              MAXPER LIT '8',
              BEGINCM LIT '0F400H',
              INTVECTOR LIT '03000H',
              CPTR LIT '1',
              FIRSTMN LIT '0',
              LASTMN LIT 'FIRSTMN+7',
              EX LIT 'FIRSTMN',

              /* MAX NUMBER OF COMPUTERS */
              /* MAX NUMBER OF MODULES IN 1 CPTR */
              /* MAX NUMBER OF MODULES IN SYSTEM */
              /* LENGTH OF MSG BUFFER */
              /* MAX NUMBER OF PRIORITY CALLS */
              /* MAX NUMBER OF PERIODIC CALLS */
              /* BEGIN OF COMMON MEMORY */
              /* INTERRUPT VECTOR */
              /* NUMBER OF HOST COMPUTER */
              /* MN OF FIRST MODULE IN CPTR */
              /* MN OF LAST MODULE IN CPTR */
              /* MN OF EXEC MODULE */

```

```

MSG$TBL$ADR LIT '0F660H';/* MSG. ENTRY TABLE ADDRESS */

/*
***
*/
4 1 = DCL (B1,B2,B3) BYTE PUBLIC,
      = (A1,A2,A3,A4) ADDRESS PUBLIC,
      = (BL,BU) BYTE PUBLIC AT (.A4);
/*
SYSTEM WORK VARIABLES
BU(UPPER) AND BL(LOWER) OVERLAY ADDRESS VARIABLE A4
*/
5 1 = DCL (ADRMSG,ADRMSGDATA) ADDRESS PUBLIC,
      = (MSG BASED ADRMSG) (4) BYTE,
      = (MSGDATA BASED ADRMSGDATA) (100) BYTE;
/*
WORK AREAS FOR MSG CONTROL BLOCK (MSG) AND MSG
DATA BYTES (MSG DATA)
ADRMSG AND ADRMSGDATA ARE SET BY EXEC BEFORE THE
CALL OF A MODULES MSG HANDLER
*/
6 1 = DCL INTMASK BYTE PUBLIC;
/*
MASK OF CURRENTLY ACTIVATED INTERRUPTS
*/
7 1 = DCL CPTR$ID BYTE PUBLIC ;

```



```

= = = = =
= = = = =
= = = = =
= = = = =
= = = = =
= = = = =
= = = = =
= = = = =
= = = = =
= = = = =
11 1 = DCL ADRSTAT ADDRESS PUBLIC,
      /*
      (MODSTAT BASED ADRSTAT) (MAXMOD) BYTE;
      /*
      MODSTAT IS AN OVERLAY FOR MODSTATUS AND COVERS
      THE ENTRIES FOR ONE COMPUTER.
      THE BASE VARIABLE ADRSTAT IS SET AT START.
      /*
12 1 = DCL BEGABSDATA BYTE PUBLIC AT (BEGINCM+MAXSYSMOD),
      /*
      BEGIN OF ABSOLUTE DATA IN COMMON MEMORY
      /*
      (RTC0,RTC1,RTC2,RTC3) BYTE PUBLIC AT (.BEGABSDATA),
      RTC (4) BYTE PUBLIC AT (.BEGABSDATA),
      CLOCK BYTE PUBLIC AT (.BEGABSDATA + 4),
      /*
      SYSTEM REALTIME CLOCK
      LEAST SIGNIFICANT BYTE = RTC(3)
      LEAST SIGNIFICANT BIT : 1 MSEC
      /*
      SYSMN10(4) BYTE PUBLIC AT (.BEGABSDATA + 5),
      SYSMN11(4) BYTE PUBLIC AT (.BEGABSDATA + 9),
      SYSMN12(4) BYTE PUBLIC AT (.BEGABSDATA + 13),
      SYSMN13(4) BYTE PUBLIC AT (.BEGABSDATA + 17),
      SYSMN14(4) BYTE PUBLIC AT (.BEGABSDATA + 21),
      SYSMN15(4) BYTE PUBLIC AT (.BEGABSDATA + 25),
      SYSMN16(4) BYTE PUBLIC AT (.BEGABSDATA + 29),

```

```

= = = = =
SYSMN17(4) BYTE PUBLIC AT (.BEGABSDATA + 33),
SYSMN18(4) BYTE PUBLIC AT (.BEGABSDATA + 37),
SYSMN19(4) BYTE PUBLIC AT (.BEGABSDATA + 41),
SYSMN23(4) BYTE PUBLIC AT (.BEGABSDATA + 45),
SYSMN24(4) BYTE PUBLIC AT (.BEGABSDATA + 49),
SYSMN25(4) BYTE PUBLIC AT (.BEGABSDATA + 53),
/* SYSTEM CONSOLE MESSAGES */
EXTMSGLOCK BYTE PUBLIC AT (.BEGABSDATA + 57),
/*
Ø IF EXT. MSG BUFFER UNLOCKED
OTHERWISE NUMBER OF CPTR CURRENTLY WORKING IN BUFFER
*/
EXTMSG BYTE PUBLIC AT (.BEGABSDATA + 58),
/*
Ø IF NO EXTERNAL MSG TO PROCESS
OTHERWISE NUMBER OF RECEIVING CPTR
*/
EXTMSGIN BYTE PUBLIC AT (.BEGABSDATA + 59),
EXTMSGOUT BYTE PUBLIC AT (.BEGABSDATA + 60),
/*
POINTER TO EXTERNAL MSG BUFFER */
NUMEXTMSG BYTE PUBLIC AT (.BEGABSDATA + 61),
/*
NUMBER OF EXT. MSG IN EXT. MSG BUFFER
*/
EXTLASTMSG BYTE PUBLIC AT (.BEGABSDATA + 62),
EXTMSGGBUFFER (512) BYTE PUBLIC AT (.BEGABSDATA + 63),
/*
EXTERNAL MESSAGE BUFFER
*/
ENDABSDATA BYTE PUBLIC AT (.BEGABSDATA + 575);
/*
END OF ABSOLUTE DATA IN COMMON MEMORY
*/
13 1 = DCL LOADSBC BYTE PUBLIC AT (MSG$TBL$ADR - 4),
      START(3) BYTE PUBLIC AT (MSG$TBL$ADR - 3 );

```

```

=
=
=
=
14 1 = DCL SAVESTACKPTR ADDRESS PUBLIC;
      /*
      ABSOLUTE LOCATED VARIABLES USED FOR LOADING AND
      START OF SBC'S
      */
=
=
=
=
15 1 = DCL CDCADR ADDRESS PUBLIC;
      /*
      STACK POINTER AT SYSTEM START
      */
=
=
=
=
16 1 = DCL EXCLEARBEG BYTE PUBLIC;
      /*
      ADDRESS TO BE CALLED AT CDC INTERRUPT
      */
=
=
=
=
17 1 = DCL (USART$IN,USART$OUT) BYTE PUBLIC;
18 1 = DCL CDCACTIVE BYTE PUBLIC;
      /* TRUE IF CDC INT ACTIVATED */
19 1 = DCL CODESAVE ADDRESS PUBLIC;
      /*
      BEGIN OF DATA TO BE CLEARED AT START
      */
=
=
=
=
20 1 = DCL MSGBUFFER (MSGBUFLN) BYTE PUBLIC,
      (MSGIN,MSGOUT) ADDRESS PUBLIC,
      NUMMSG BYTE PUBLIC,
      LASTMSG ADDRESS PUBLIC;
      /*
      MSGBUFFER IS A CIRCULAR FIFO LIST CONTAINING THE
      MSG WHICH ARE WAITING TO BE PROCESSED.
      MSGIN POINTS TO LOCATION OF NEXT MSG TO FILL IN.
      MSGOUT POINTS TO MSG TO BE PROCESSED NEXT.
      NUMMSG IS INCREMENTED WHEN A MSG IS SENT AND
      DECREMENTED WHEN A MSG IS PROCESSED.
      NUMMSG = 0: MSGBUFFER IS EMPTY.
      LASTMSG CONTAINS INDEX OF LAST BYTE OF LAST MSG

```

```

=
=
21 1 =      IN MSGBUFFER + 1.
=      /*
=      DCL PRIORLIST (MAXPRIOR) ADDRESS PUBLIC,
=      PRIORSCHEDULE BYTE PUBLIC;
=      /*
=      PRIORLIST CONTAINS THE ADDRESSES OF THE ACTIVATED
=      PRIORITY TASKS.
=      A PRIORITY TASK IS CALLED WHEN IT IS SCHEDULED BY
=      SETTING THE RESPECTIVE BIT IN PRIORSCHEDULE,
=      E.G.: IF BIT 3 (LSB = BIT 0) IN PRIORSCHEDULE IS
=      SET, THE ADDRESS IN PRIORLIST(3) IS CALLED.
=      /*
22 1 =      DCL (XB1,XB2,XB3) BYTE PUBLIC, (XA1,XA2,XA3) ADDRESS PUBLIC;
=      /*
=      EXEC WORK VARIABLES
=      /*
23 1 =      DCL PERXTBL (MAXPER) BYTE PUBLIC;
=      /*
=      PERXTBL CONTAINS POINTER TO PERLIST IN COMPACT FORM
=      0FFH - PERIODIC NOT ACTIVATED
=      /*
24 1 =      DCL PERLIST (MAXPER) STRUCTURE (
=      FREE BYTE,          /* TRUE - ITEM IS FREE */
=      PERADR ADDRESS,      /* ADDRESS OF PERIODIC */
=      PERINTADR ADDRESS,   /* ADDRESS OF TIME INTERVAL */
=      PERTIME (4) BYTE) PUBLIC, /* NEXT CALL TIME (RTC FORMAT) */
=      /*
=      PERLIST CONTAINS ALL SIGNIFICANT DATA OF AN
=      ACTIVATED PERIODIC TASK
=      /*
=      (PERINT BASED XA1) (5) BYTE,
=      /*
=      OVERLAY FOR TIME INTERVAL OF A PERIODIC
=      /*
=      PERX BYTE PUBLIC,
=      /*

```

10 10 10 10 10 10 10 10 10 10 10 10 10


```

26      $ EJECT
27      $ INCLUDE(:F1:SCEXT.SRC)
28      /*****
29      *****/
30      *****/
31      *****/
32      *****/
33      *****/
34      *****/
35      *****/
36      *****/
37      *****/
38      *****/
39      *****/
40      *****/
41      *****/
42      *****/
43      *****/
44      *****/
45      *****/
46      *****/
47      *****/
48      *****/
49      *****/
50      *****/
51      *****/

: EXTERNAL DECLARATIONS OF SYSTEM ROUTINES

*****/

FILL: PROCEDURE (P1,P2,P3) EXTERNAL;
      DCL (P1,P2) ADDRESS, P3 BYTE;
      END;

CLEARDATA: PROCEDURE (P1,P2) EXTERNAL;
      DCL (P1,P2) ADDRESS;
      END;

SHFT: PROCEDURE (P1) BYTE EXTERNAL;
      DCL P1 BYTE;
      END;

BIT : PROCEDURE (P1,P2) BYTE EXTERNAL;
      DCL (P1,P2) BYTE;
      END;

CLEARBIT: PROCEDURE (P1,P2) EXTERNAL;
      DCL P1 BYTE, P2 ADDRESS;
      END;

SETBIT: PROCEDURE (P1,P2) EXTERNAL;
      DCL P1 BYTE, P2 ADDRESS;
      END;

LEGAL: PROCEDURE (P1,P2,P3) BYTE EXTERNAL;
      DCL (P1,P2,P3) BYTE;
      END;

SETPERTIME: PROCEDURE (P1) EXTERNAL;
      DCL P1 BYTE;
      END;

COPY$CLOCK: PROCEDURE (P1) EXTERNAL;
      DCL P1 ADDRESS;

```



```

52 = 2
53 = 1
54 = 2
55 = 2
56 = 1
57 = 2
58 = 2
59 = 1
60 = 2
61 = 2
62 = 1
63 = 2
64 = 2

    TIME$CMP : PROCEDURE (P1,P2) BYTE EXTERNAL;
    DCL (P1,P2) ADDRESS;
    END;
    PER$ACT : PROCEDURE (P1,P2) BYTE EXTERNAL;
    DCL (P1,P2) ADDRESS;
    END;
    PER$CHG : PROCEDURE (P1,P2) EXTERNAL;
    DCL P1 BYTE, P2 ADDRESS;
    END;
    PER$SUSP : PROCEDURE (P1) EXTERNAL;
    DCL P1 BYTE;
    END;

MSG$OVERFLOW: PROCEDURE EXTERNAL;
END;
PACK$MCB:PROCEDURE (P1) EXTERNAL;
DCL P1 ADDRESS;
END;
SEND :PROCEDURE (P1) BYTE EXTERNAL;
DCL P1 ADDRESS;
END;
GET :PROCEDURE (P1) EXTERNAL;
DCL P1 ADDRESS;
END;
RELEASE:PROCEDURE (P1) EXTERNAL;
DCL P1 ADDRESS;
END;
SET$EXT$MSG:PROCEDURE EXTERNAL;
END;
REC$EXT:PROCEDURE EXTERNAL;
END;
SEND$EXT:PROCEDURE EXTERNAL;
END;
ILLEGAL$MSG:PROCEDURE EXTERNAL;
END;

```

```

87 1 = PRIORITY:PROCEDURE (P1) EXTERNAL;
88 2 = DCL P1 BYTE;
89 2 = END;
90 1 = ENTER$PRIOR: PROCEDURE (P1) BYTE EXTERNAL;
91 2 = DCL P1 ADDRESS;
92 2 = END;
93 1 = REM$PRIOR: PROCEDURE (P1) EXTERNAL;
94 2 = DCL P1 BYTE;
95 2 = END;
96 1 = SET$CDC:PROCEDURE (P1,P2) BYTE EXTERNAL;
97 2 = DCL (P1,P2) ADDRESS;
98 2 = END;
99 1 = ENTER$INT: PROCEDURE (P1) EXTERNAL;
100 2 = DCL P1 BYTE;
101 2 = END;
102 1 = REM$INT:PROCEDURE (P1) EXTERNAL;
103 2 = DCL P1 BYTE;
104 2 = END;
105 1 = PRINT$ASYNC:PROCEDURE (P1,P2) EXTERNAL;
106 2 = DCL P1 ADDRESS, P2 BYTE;
107 2 = END;
108 1 = PRINT$SYNC:PROCEDURE (P1,P2,P3) EXTERNAL;
109 2 = DCL P1 ADDRESS, (P2,P3) BYTE;
110 2 = END;
111 1 = CON$INPUT$REQ:PROCEDURE (P1,P2) EXTERNAL;
112 2 = DCL (P1,P2) BYTE;
113 2 = END;
114 1 = CON$LINK$REQ:PROCEDURE (P1) EXTERNAL;
115 2 = DCL P1 BYTE;
116 2 = END;
117 1 = PAR$LINK$REQ:PROCEDURE (P1) EXTERNAL;
118 2 = DCL P1 BYTE;
119 2 = END;
120 1 = CON$P$LEASE: PROCEDURE EXTERNAL;

```

```

121 =
122 =
123 =
124 =
125 =
126 =
127 =
128 =
129 =
130 =
131 =
132 =
133 =
134 =
135 =
136 =
137 =
138 =
139 =
140 =
141 =
142 =
143 =
144 =
145 =
146 =
147 =
148 =
149 =
150 =
151 =
152 =
153 =
154 =
155 =

END;
CON$NEW$LINE:PROCEDURE (P1) EXTERNAL;
    DCL P1 BYTE;
END;
CON$BEG$LINE:PROCEDURE (P1) EXTERNAL;
    DCL P1 BYTE;
END;
PRINT$LINKED:PROCEDURE (P1,P2,P3,P4) EXTERNAL;
    DCL P1 ADDRESS, (P2,P3,P4) BYTE;
END;
PAR$RELEASE: PROCEDURE EXTERNAL;
END;
WRITE: PROCEDURE (P1,P2,P3) EXTERNAL;
    DCL P1 ADDRESS, (P2,P3) BYTE;
END;
WRITE$LINKED: PROCEDURE (P1,P2,P3,P4) EXTERNAL;
    DCL P1 ADDRESS, (P2,P3,P4) BYTE;
END;
PAGE: PROCEDURE (P1) EXTERNAL;
    DCL P1 BYTE;
END;

ACTIVE: PROCEDURE (P1) BYTE EXTERNAL;
    DCL P1 BYTE;
END;
UPD$STAT:PROCEDURE (P1,P2) EXTERNAL;
    DCL (P1,P2) BYTE;
END;
CONV$ASC:PROCEDURE (P1) EXTERNAL;
    DCL P1 BYTE;
END;
DEBLK : PROCEDURE EXTERNAL;
END;
GET$NUM:PROCEDURE BYTE EXTERNAL;
END;
VECTOR$ADD: PROCEDURE (P1,P2,P3) EXTERNAL;

```

```

156 2 = DCL (P1,P2,P3) ADDRESS;
157 2 = END;
158 1 = VECTOR$SUB: PROCEDURE (P1,P2,P3) EXTERNAL;
159 2 = DCL (P1,P2,P3) ADDRESS;
160 2 = END;
161 1 = NUMOUT: PROCEDURE (P1,P2,P3,P4,P5) EXTERNAL;
162 2 = DCL (P1,P4) ADDRESS, (P2,P3,P5) BYTE;
163 2 = END;
164 1 = GET$CHAR: PROCEDURE BYTE EXTERNAL;
165 2 = END;
=
166 1 = ENTER$MSG$MOD: PROCEDURE (P1,P2) EXTERNAL;
167 2 = DCL P1 BYTE, P2 ADDRESS;
168 2 = END;
169 1 = MONITOR: PROCEDURE EXTERNAL;
170 2 = END;
171 1 = ERROR : PROCEDURE (P1,P2,P3) EXTERNAL;
172 2 = DCL P1 ADDRESS, (P2,P3) BYTE;
173 2 = END;

```

```

174 1      $EJECT
175 2      EXSTART:  PROCEDURE EXTERNAL;
176 1      DCL SYSSTART LABEL PUBLIC; /* ENTRY POINT FOR RESTART PROCEDURE */
/*
*****
**
**
EXECUTIVE
**
*****
*/
177 1      EXEC:      PROCEDURE PUBLIC;
178 2      DCL TMP$CLOCK (4) BYTE; /* TEMPORARY CLOCK VALUE */
179 2      EXEC0:     CALL EXSTART;
/* INITIALIZE */
/*
*****
**
**
PROCESS PRIORITY TASKS OF MODULES
*/
180 2      EXEC1:     DO WHILE PRIORSCHEDULE <> 0;
181 3      /* DO AS LONG AS PRIORITY CALLS ARE SCHEDULED */
182 3      XB2 = 1;
183 4      DO XB1 = 1 TO MAXPRIOR;
184 4      /* FIND HIGHEST PRIORITY SCHEDULED */
185 4      XB3 = SCR(PRIORSCHEDULE,XB1);
186 5      IF CARRY THEN
187 5      DO;
188 5      PRIORSCHEDULE = PRIORSCHEDULE XOR XB2;
189 5      /* RESET PRIORITY BIT */
190 5      XA1 = PRIORLIST(XB1-1);

```



```

188 5      CALL XA1;
189 5      /* CALL PRIORITY PROCEDURE */
190 5      GO TO EXEC11;
191 4      END;
192 4      XB2 = ROL(XB2,1);
193 3      END;
194 3      EXEC11:
          ;
          END;
/*
*****
**
          PROCESS PENDING REAL TIME MESSAGES
*/
195 2      EXEC2:      IF NUMMSG = 0 THEN GO TO EXEC21;
197 2      /* NO MSG TO PROCESS */
          IF MSGOUT = LASTMSG THEN LASTMSG,MSGOUT = 0;
          /* NEXT MSG AT TOP OF MSGBUFFER */
199 2      XA1 = MSGOUT;
200 2      MSGOUT = XA1 + MSGBUFFER(XA1+ML);
201 2      /* COMPUTE BEGIN OF NEXT MSG */
          NUMMSG = NUMMSG - 1;
          /* UPDATE NUMBER OF MSG */
202 2      ADRMSG = .MSGBUFFER(XA1);
203 2      ADRMSGDATA = ADRMSG + 4;
204 2      /* 'SET' MSG INTO WORKAREA */
          IF MSG(RMN) <= LASTMN AND
             MSG(RMN) >= FIRSTMN THEN DO;
             XA1 = MSG$MOD$ADDRESS(MSG(RMN));
             CALL XA1;
          END;
206 3      ELSE CALL SENDEXT;
207 3      GO TO EXEC1;
208 3
209 2
210 2
211 2      EXEC21:      /* CHECK PRIORITY CALLS AGAIN */
          IF EXTMSG <> CPTR THEN GOTO EXEC3;
          /* NO EXTERNAL MSG TO PROCESS */
213 2      CALL REEXT;

```



```

214 2      /* TRANSFER MSG INTO OWN MSG BUFFER */
      GO TO EXEC1;
      /* CHECK PRIORITY CALLS AGAIN */
      /* *****
      **
      */
      CHECK FOR PERIODIC CALLS OF MODULES
EXEC3:
215 2      ;
216 2      IF NUMBER = 0 THEN GO TO EXEC4;
      /* NO PERIODIC CALLS ACTIVE */
218 2      XB1 = NUMBER - 1;
      /* SET LIMIT FOR 1 SEARCH CYCLE */
219 2      CALL COPY$CLOCK(.TMP$CLOCK);
      /* COPY CLOCK VALUE */
220 2      DO XB2 = 0 TO XB1;
      /* CHECK ALL ACTIVATED PERIODICS */
221 3      PERX = PERXTBL(XB2);
222 3      IF TIME$CMP(.PERLIST(PERX).PERTIME,.TMP$CLOCK) THEN
      /* CALL TIME REACHED */
223 3      DO;
224 4      XA1 = PERLIST(PERX).PERADR;
225 4      CALL XA1;
      /* CALL PERIODIC */
226 4      CALL SETPERTIME(PERX); /* SET NEXT CALL TIME */
227 4      GO TO EXEC1;
      /* CHECK PRIORITY CALLS */
228 4      END;
229 3      END;
      /* *****
      **
      */
      BACKGROUND TASKS
      /*
EXEC4:
230 2      ;
231 2      OUTPUT(0D6H) = TRUE; /* LIGHT LED FOR ONE MILISECOND */

```

GO TO EXEC1;
RETURN;
END EXEC;

232 2
233 2
234 2

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE INTMSG
 NO OBJECT MODULE REQUESTED
 COMPILER INVOKED BY: PLMS0 :F1:INTMSG.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)

```

1      INTMSG: DO;
      $ NOLIST
      $ INCLUDE(:F1:SEIODE.SRC)
      /*****
      *****
      : EXTERNAL DECLARATIONS FOR PUBLIC
      SERIAL I/O INTERFACE PROCEDURES.
      *****/
      */

174      SEIO$START: PROCEDURE EXTERNAL;
175      END;
176      SE$PRINT$ASYNC: PROCEDURE EXTERNAL;
177      END;
178      SE$PRINT$SYNC : PROCEDURE EXTERNAL;
179      END;
180      SE$INPUT$REQ: PROCEDURE EXTERNAL;
181      END;
182      SE$INP$ACTIVATE:PROCEDURE EXTERNAL;
183      END;
184      TERMIO: PROCEDURE EXTERNAL;
185      END;
186      SE$NEW$LINE: PROCEDURE EXTERNAL;
187      END;
188      SE$BEG$LINE: PROCEDURE EXTERNAL;
189      END;
190      SE$PRINT$LINKED:PROCEDURE EXTERNAL;
191      END;

```

```

= = = = =
$EJECT
$ INCLUDE(:F1:PAIODE.SRC)
/*****
*****
*****
: EXTERNAL DECLARATIONS OF PUBLIC PARALLEL
  OUTPUT INTERFACE PROCEDURES.
*****
***/

= = = = =
DCL PRELL$OUT BYTE EXTERNAL;

= = = = =
PAIO$START: PROCEDURE EXTERNAL;
1          END;
193
194
PA$OUT$ACTIVATE:PROCEDURE EXTERNAL;
1          END;
195
196
PA$OUT$RELEASE: PROCEDURE EXTERNAL;
1          END;
197
198
PA$PRINT: PROCEDURE EXTERNAL;
1          END;
199
200
PA$NEW$PAGE: PROCEDURE EXTERNAL;
1          END;
201
202
PA$PRINT$LINKED:PROCEDURE EXTERNAL;
1          END;
203
204

= = = = =
$ INCLUDE(:F1:MONIDE.SRC)
/*****
*****
*****
: EXTERNAL DECLARATIONS OF MONITOR
  PUBLIC PROCEDURES.
*****
***/

= = = = =
MONITOR$TRAP:PROCEDURE EXTERNAL;
1          END;
205
206
MONI$INTER: PROCEDURE EXTERNAL;
1          END;
207
208
MONI$INIT: PROCEDURE EXTERNAL;
1          END;
209
```

210	2	=	END;
211	1	=	MONI\$ENDER: PROCEDURE EXTERNAL;
212	2	=	END;
213	1	=	PRINT\$LINE: PROCEDURE EXTERNAL;
214	2	=	END;


```

$EJECT
/*
*****
**
**
**
**
**
*****
      INTERRUPT PROCESSING
*****
/*
*****
**
**
**
**
*****
215 1  INTRESET:  PROCEDURE;
216      OUTPUT(0DAH) = 00100000B;
217      RETURN;
218      END INTRESET;
/*
*****
**
**
**
**
*****
219 1  INT0:      PROCEDURE INTERRUPT 0;
                /* CDC INTERRUPT */

                IF NOT CDCACTIVE THEN RETURN;
                OUTPUT(0DFH) = 01110000B; /* CNT 1 MODE 0 */
                CALL CDCADR;
                /* CALL PROCESS OF INTERRUPT */
                INTMASK = INTMASK OR 1;
                OUTPUT(0DBH) = INTMASK;
                /* DISABLE INT0 */
                CALL INTRESET;
                CDCACTIVE = FALSE;
                RETURN;
                END INT0;
/*
*****
**
*****
230 1  INT1:      PROCEDURE INTERRUPT 1;

```

```

231 2      CALL MONITOR$TRAP;
232 2      CALL INTRESET;
233 2      RETURN;
234 2      END INT1;
/*
*****
*/
235 1      PROCEDURE INTERRUPT 2;
236 2      DCL INCR BYTE DATA(1);
/* RTC INTERRUPT */

237 2      IF CLOCK = 0 THEN DO;
239 3          CLOCK = 0FFH;
240 3          RTC3 = RTC3 + INCR ;
241 3          RTC2 = RTC2 PLUS 0 ;
242 3          RTC1 = RTC1 PLUS 0 ;
243 3          RTC0 = RTC0 PLUS 0 ;
244 3          CLOCK = 0;
245 3          /* UPDATE RTC */
246 2          OUTPUT(0DCH) = 33H;
247 2          OUTPUT(0DCH) = 4;
/* 930 NS X 433H (1075) = .99975 MS */
/* SET CTR 0 TO 1 MSEC IN MODE 0 */

248 2      CALL INTRESET;
249 2      RETURN;
250 2      END INT2;
/*
*****
*/
251 1      PROCEDURE INTERRUPT 3;

```

```

252 2      CALL PRIORITY(USART$IN);
253 2      /* SCHEDULE PRIORITY CALL */
254 2      CALL INTRESET;
255 2      RETURN;
      END INT3;
/*
*****
*/
256 1      PROCEDURE INTERRUPT 4;
      INT4:
257 2      CALL PRIORITY(USART$OUT);
258 2      /* SCHEDULE PRIORITY CALL */
259 2      CALL INTRESET;
260 2      RETURN;
      END INT4;
/*
*****
*/
261 1      PROCEDURE INTERRUPT 5;
      INT5:
262 2      CALL PRIORITY(PRELL$OUT);
263 2      CALL INTRESET;
264 2      RETURN;
265 2      END INT5;
/*
*****
*/
266 1      DCL SYSSTART LABEL EXTERNAL; /* RESTART ENTRY POINT */
267 1      EXEC:      PROCEDURE EXTERNAL;
268 2      END EXEC;
269 1      INT6:      PROCEDURE; /* RESTART INTERRUPT */

```

270	2	STACKPTR = SAVESTACKPTR;
271	2	/* RESET STACK POINTER */
272	2	LOADSBC = 1; /* COMPUTER 1 BEGINS PROCESS */
		GO TO SYSSTART;
273	2	END INT6;

\$ EJECT

SYSTEM MESSAGE TASK

RECEIVES REQUEST MESSAGES FOR THE USE OF THE I/O FACILITIES AND THE MONITOR.

[illegible]

MSGENT0: PROCEDURE PUBLIC;

```
IF MSG(MN) < 10 OR MSG(MN) > 26 /* CHECK BOUNDS */
```

THEN CALL ILLEGAL\$MSG;

```
ELSE DO CASE MSG(MN)-10 ; /* SWITCH TO CORRESPONDING ACTION */
```

```

CALL SE$PRINT$(CNC ; /* MN 10 *
CALL SE$PRINT$(SYNC ; /* MN 11 *
CALL SE$INPUT$(REQ ; /* MN 12 *
CALL SE$INPUT$(ACTIVATE; /* MN 13 *
CALL PA$OUT$(ACTIVATE; /* MN 14 *
CALL TERMIO ; /* MN 15 *
CALL SE$NEW$(LINE ; /* MN 16 *
CALL SE$BEG$(LINE ; /* MN 17 *
CALL SE$PRINT$(LINKED; /* MN 18 *
CALL PA$OUT$(RELEASE ; /* MN 19 *
CALL MONI$(INTER ; /* MN 20 *
CALL MONI$(INIT ; /* MN 21 *
CALL MONI$(ENDER ; /* MN 22 *
CALL PA$PRINT ; /* MN 23 *
CALL PA$NEW$(PAGE ; /* MN 24 *
CALL PA$PRINT$(LINKED; /* MN 25 *
CALL PRINT$(LINE ; /* MN 26 *
END;

```

RETURN;

END MSGENT0;

```

$ EJECT
/*
*****
SETS EXECUTIVE DATA STRUCTURES.
*/
SET$EX$DATA: PROCEDURE;
  DCL ERM(7) BYTE DATA('ERROR: ');

  CALL CLEAR$DATA(.EX$CLEAR$BEG,.EX$CLEAR$END);
  CALL MOVE(7,.ERM,.ERRORMSG);
  CALL ENTER$MSG$MOD(EX,.MSGENT0);
  CALL UPDSTAT(EX,0BH); /* ACTIVATE EXEC. MSG. HANDLER */
  DO B1 = 0 TO LAST(PERLIST); /* INIT PERIODIC LIST */
    PERLIST(B1).FREE = TRUE;
  END;
  ADRSTAT = .MODSTATUS(0) + FIRSTMN ; /* SET OVERLAY */
  IF CPTR = 1 THEN CALL CLEAR$DATA(.BEGABSDATA,.ENDABSDATA);
  RETURN;
END;

```

```

298 1
299 2
300 2
301 2
302 2
303 2
304 2
305 3
306 3
307 2
308 2
310 2
311 2

```



```

312 $ EJECT
313 /*
314 *****
315 *****
318 *****
321 *****
324 *****
327 *****
328 *****
329 *****
330 *****
333 *****
334 *****
336 *****
337 *****
338 *****
339 *****
340 *****
341 *****
342 *****

SETS INITIAL SYSTEM MESSAGE STATES
*/
SET$EX$MSG: PROCEDURE;
  DCL SYSMN0(4) BYTE ; /* START MESSAGE */
  NUMMSG = 0 ; /* INIT NUMBER OF MESSAGES FLAG */
  SYSMN10(2)=10;SYSMN11(2)=11;SYSMN12(2)=12;
  SYSMN13(2)=13;SYSMN14(2)=14;SYSMN15(2)=15;
  SYSMN16(2)=16;SYSMN17(2)=17;SYSMN18(2)=18;
  SYSMN19(2)=19;SYSMN23(2)=23;SYSMN24(2)=24;
  SYSMN25(2)=25;
  SYSMN12(3)=5;
  SYSMN13(3),SYSMN14(3),SYSMN15(3),
  SYSMN16(3),SYSMN17(3),SYSMN19(3),SYSMN24(3)=4;
  SYSMN0(1)=EX;SYSMN0(2)=0;SYSMN0(3)=4;
  DO B1 = 1 TO LAST(MODSTAT);/* SEND INIT MSG IF PRESENT */
    IF MODSTAT(B1) <> 0 THEN DO;
      SYSMN0(0) = B1 + FIRSTMN;
      CALL PACKMCB(.SYSMN0);
      NUMMSG = NUMMSG + 1 ;
    END;
  END;
  RETURN;
END;

```

```

$ EJECT
/*
*****
SETS INITIAL INTERRUPT STATE
*/
SET$EX$INTE: PROCEDURE;
/* INITIALIZE INTERRUPT VECTOR */
INT0$LOC,INT1$LOC,INT2$LOC,INT3$LOC,
INT4$LOC,INT5$LOC,INT6$LOC,INT7$LOC = 0C3H; /* JMP */
INT0$PRO = .INT0 ; INT1$PRO = .INT1 ;
INT2$PRO = .INT2 ; INT3$PRO = .INT3 ;
INT4$PRO = .INT4 ; INT5$PRO = .INT5 ;
INT6$PRO = .INT6 ; INT7$PRO = .INT6 ;
A4 = INTVECTOR ;
B1 = BL AND 11100000B;
OUTPUT(0DAH) = B1 OR 00010010B; /* ICW1 */
OUTPUT(0DBH) = BU ; /* ICW2 */
INTMASK = 10111111B ; /* ONLY RESET INTERRUPT SET */
RETURN;
END;

```

```

343 1
344 2
345 2
347 2
349 2
351 2
353 2
354 2
355 2
356 2
357 2
358 2
359 2

```

360 \$ EJFCT

361 /*

362 *****

363 **

364 START

INITIALIZE COMPUTER

365 **

366 *****

367 */

368 EXSTART: PROCEDURE PUBLIC;

369 DCL SYS\$TAG(*) BYTE DATA('RDY COMPUTER', CPTR+30H, 0AH, 0DH);

370 DISABLE;

371 CALL SET\$EX\$DATA;

372 CALL SET\$EX\$MSG;

373 CALL SET\$EX\$INTE;

374 OUTPUT(0DFH) = 00110000B; /* CNT 0 MODE 0 */

375 OUTPUT(0DFH) = 01110000B; /* CNT 1 MODE 0 */

376 IF CPTR = 1 THEN

377 DO;

378 INTMASK = INTMASK XOR 00000100B; /* RTC INTERR. */

379 OUTPUT(0DCH) = 33H;

380 OUTPUT(0DCH) = 4; /* LOAD COUNTER 0 WITH 1 MSEC 433H */

381 END;

382 LOADSBC = LOADSBC + 1; /* TRIGGER NEXT COMPUTER */

383 CALL SEIO\$START; /* INITIALIZE SERIAL IO MODULE */

384 CALL PAIO\$START; /* INITIALIZE PARALLEL OUTPUT MODULE */

385 OUTPUT(0DBH) = INTMASK; /* INIT INTERRUPT MASK */

386 CALL PRINT\$ASYNC(.SYS\$TAG, LENGTH(SYS\$TAG));

387 ENABLE;

388 RETURN;

389 2

PL/M-80 COMPILER

PAGE 14

381 2 END EXSTART;
382 1 END INTMSG;

MODULE INFORMATION:

CODE AREA SIZE	= 0382H	898D
VARIABLE AREA SIZE	= 0004H	4D
MAXIMUM STACK SIZE	= 000AH	10D
627 LINES READ		
0 PROGRAM ERROR(S)		

END OF PL/M-80 COMPILATION

SE\$IO\$MOD: DO;
\$NOLIST

SERIAL IO DATA

CRT I/O PORTS AND USART CONTROL WORDS

```

DCL CRTDATAIN LIT '0FCH',
CRTSTATIN LIT '0EDH',
CRTDATAOUT LIT '0ECH',
CRTCMDOUT LIT '0EDH',
TMCP LIT '0DFH',
CTR2 LIT '0DEH',
B2400 LIT '01CH',
MCW LIT '01001110B',
CCW1 LIT '00100111B',
CCW2 LIT '00100110B',
RESET LIT '01000000B',

/* DATA INPUT */
/* STATUS INPUT */
/* DATA OUTPUT */
/* COMMAND OUTPUT */
/* TIMEE COMM. PORT */
/* COUNTER 2 PORT */

/* MODE CONTROL WORD */
/* COMMAND CONTROL WORD */
/* COMMAND CONTROL WORD 2 */
/* RESET CRT */

```

CRT CONTROL CHARACTERS

DCL HOME LIT '2',	/* HOME CURSOR */
FC LIT '1CH',	/* FORWARD CURSOR
BC LIT '8',	/* BACK CURSOR */


```

UC LIT '1AH',
CR LIT '0DH',
LF LIT '0AH',
CLEAR LIT '1EH',
ADRY LIT '0CH',
POLL LIT '1DH',
BELL LIT '7',
CONSOL$ID LIT '15H',
MONITOR$ID LIT '14H',
CLRLINE LIT '17H',
BOL LIT '96',
ASYNCLINE LIT '102',
INPLINE LIT '119',
ARROW LIT '5EH',
BLANK LIT ' ',
LINELENGTH LIT '79',
CRTOUTLEN LIT '256',
DELINPUT LIT '7FH';

```

```

/*
*****
**
**
**
**
*****

```

MESSAGE CONTROL BLOCKS

```

28 1 DCL CSMN20(4) BYTE , INPUT TEXT MSG */
/* TRANSFER INPUT TEXT MSG */
CSMN21(4) BYTE ,
/* INPUT REQUEST TELL BACK MSG
1ST DATA BYTE: TRUE - REQUEST ACKNOWLEDGED
FALSE OTHERWISE */
CSMN22(4) BYTE ,
/* TERMINATE I/O MSG
SEND TO MODULE CONNECTED TO CRT IF INPUT IS 'INT' */
CSMN26(4) BYTE ;
/* TELLBACK MSG */

```

```

/*
*****
**
**
**
**
*****

```



```

**
**
**/
29 1 DCL ROLLSCH (8) BYTE DATA (CR,ADXY,BOL,ASYNCLINE+1,CLRLINE,ADXY,BOL,
/* ROLL SCREEN,CLEAR LINE AFTER ASYNCLINE,RETURN TO
    INPLINE),
    BEGIN OF INPUT LINE */
    NEWL(2) BYTE DATA (1,','),
    BEGL(4) BYTE DATA (0,ADXY,BOL,INPLINE),
    ASYNC1(10) BYTE DATA (ARROW,ADXY,BOL,ASYNCLINE,CLRLINE,
    BELL,'***');
/* START ASYNCHRONOUS PRINT */
/* *****
**
**
**/
VARIABLE DATA
30 1 DCL CSBEGDATA BYTE,
    INACTMOD BYTE,
/* MODULE WITH CURRENT INPUT ACTIVE */
    CSTBCODE BYTE, /* REQUESTED TELLBACK CODE */
    CRTIN(LINELENGTH) BYTE,
/* CRT INPUT BUFFER */
    CRTOUT(CRTOUTLEN) BYTE,
/* CRT OUTPUT BUFFER */
    (INX,OUTX) BYTE,
/* POINTER TO CRTIN AND CRTOUT (NEXT TO FILL) */
    (CURSORINP,CURSORSAVE) BYTE,
/* CURSOR POSITION ON INPUT LINE */
    OUTACTIVE BYTE,
/* TRUE IF OUTPUT ACTIVE */
    INPREQUEST BYTE,
/* TRUE IF INPUT REQUEST MSG RECEIVED */
    OUTPTR BYTE,
/* POINTER TO NEXT CHARACTER TO PRINT */
    ROLLSCREEN BYTE,

```

```
/* TRUE IF SCREEN IS TO ROLL AFTER INPUT */
N$OF$RATES BYTE,
/* NUMBER OF RATES TESTED */
BAUD$RATE ADDRESS,
/* BAUD RATE BEEN TESTED */
CHAR BYTE,
/* INPUT CHARACTER */
C$TELLBACK BYTE,
/* TRUE IF TELLBACK MSG REQUIRED AFTER SYNC OUTPUT */
C$ENDDATA BYTE;
/* END OF VARIABLE DATA */
```

✱

CSCONVXY

CONVERT INPUT TO XY-ADDRESSING

```
179 1 C$CONVY: PROCEDURE (B) BYTE;
```

130 2 DCL (A,B) BYTE;

181 2 A = B + 95;

```
182 2 IF B < 33 THEN RETURN A;
```

184 2 A = A - 64;

```
185 2 IF B < 65 THEN RETURN A;
```

187 2 $A = A - 64;$

188 RETURN A; 2

189 2 END CCONVXY;


```

262 2 IF OUTACTIVE OR NOT INPREQUEST THEN RETURN;
264 2 IF CHAR = DELINPUT THEN
265 2 /* DELETE LAST CHARACTER */
266 3 DO;
268 3 IF CURSORINP <= CURSORSAVE
269 4 THEN CALL PACKCRTOUT(BELL);
270 4 ELSE DO;
271 4 CALL PACKCRTOUT(BC);
272 4 CALL PACKCRTOUT(CRLINE);
273 4 INX = INX - 1;
274 3 CURSORINP = CURSORINP - 1;
275 3 END;
276 3 CALL STARTOUT;
277 3 RETURN;
278 3 END;
279 2 IF CHAR = CR THEN
280 2 /* END OF INPUT */
281 2 DO;
282 3 CSMN20(0) = INACTMOD;
283 3 /* SET RECEIVING MODULE */
284 3 CSMN20(3) = INX + 4;
285 3 /* MSG LENGTH */
286 3 INPREQUEST = FALSE;
287 3 IF SEND(.CSMN20(0)) THEN
288 4 DO;CRTIN(INX) = EOT;
289 4 CALL MOVE(INX,.CRTIN(1),ADMSGDATA);
290 4 END;
291 3 ELSE INACTMOD = 0FFH;
292 3 /* RECEIVING MODULE NO LONGER ACTIVE */
293 3 INX = 1;
294 3 IF ROLLSCREEN THEN
295 4 DO;
296 4 CALL PACK(.ROLLSCR, LAST(ROLLSCR));
297 4 CURSORINP, CURSORSAVE=1;

```

```

293 4      CALL STARTOUT;
294 4      END;
295 3      RETURN;
296 3      END;

297 2      IF CHAR < 20H OR CHAR > 5AH OR CURSORINP >= LAST(CRTIN)
      THEN DO; /* INVALID CHARACTER */
299 3          CALL PACKCRTOUT('?');
300 3          CALL PACKCRTOUT(BELL);
301 3          CALL PACKCRTOUT(BC);
302 3          END;
303 2      ELSE DO;
304 3          CALL PACKCRTOUT(CHAR);
305 3          CRTIN(INX) = CHAR;
306 3          INX = INX + 1;
307 3          CURSORINP = CURSORINP + 1;
308 3          END;
309 2          CALL STARTOUT;
310 2          RETURN;
311 2      END CSINPUT;

```

SUBJECT

✱

✻

SEPRINTASYN

PRINT ASYNCHRONOUS TEXT (MN 10)
(MAX 1 LINE)

**

/ *

SEPRINTASNC: PROCEDURE PUBLIC ;
DCL ASYNC2(3) BYTE;

```

ASYNC2(0) = ADRXY ; ASYNC2(2) = INPLINE ;
CALL PACK(.ASYNC1, LAST(ASYNC1));
    CALL PACK(ADRMMSGDATA, MSG(ML)-5);
/* PACK MSG INTO OUTPUT BUFFER */
ASYNC2(1) = C$CONVXY(CURSORINP);
/* RETURN CURSOR TO LAST INPUT POSITION */
CALL PACK(.ASYNC2, LAST(ASYNC2));
CALL STARTOUT;
RETURN;
END SEPRINTASYN;

```

141

SUBJECT

✱

✻

SEPRINTSYNC

```
PRINT SYNCHRONOUS TEXT (MN 11 AND MN 18)
PRINT ON INPUT LINE
IF MSGDATA(0) = TRUE : ROLL SCREEN AFTER OUTPUT
```

✱✱

—
*

SEPRINTSYNC: PROCEDURE PUBLIC;

IF NOT\$ACT THEN RETURN;

/* MSG NOT FROM CORRECT MODULE */

CALL PACK (ADMSGDATA+1,MSG(ML)-6);

/* MOVE CHARACTERS FROM MSG TO CRT OUTPUT BUFFER */

IF MSGDATA(0)

```
THEN CALL PACK(.ROLLSCR, LAST(ROLLSCR));
```

/* ROLL SCREEN AFTER OUTPUT */

```
ELSE CURSORINP = CURSORINP + MSG(ML) - 5;
```

CALL STARTOUT;

RETURN;

END SEPRINTSYNC;

143


```

$EJECT
/*
*****
**
**
          POSITION CURSOR
**
*****
**
          POS$CUR
          DCL A ADDRESS,B BYTE;
          MSG(ML) = B ;
          ADRMSGDATA = A ;
          CALL SEPRINTSYNC ;
          RETURN;
          END POS$CUR;
/*
*****
**
          SE$NEW$LINE
**
*****
**
          SE$NEW$LINE: PROCEDURE PUBLIC;
          CALL POS$CUR(.NEWL,6);
          RETURN;
          END;

```

```

370 1
371 2
372 2
373 2
374 2
375 2
376 2
377 1
378 2
379 2
380 2

```

```

$EJECT
/*
*****
**
**
**
SE$BEG$LINE
**
*****
**/
1 SE$BEG$LINE: PROCEDURE PUBLIC;
2 CALL POS$CUR(.BEGL,8);
3 CURSORINP,CURSORSAVE = 1 ;
4 RETURN;
5 END;
/*
*****
**
**
INIT$USART: INITIALIZE USART BAUD RATE
**
*****
**/
1 INIT$USART: PROCEDURE;
2 OUTPUT(CRTCMDOUT) = RESET;
3 OUTPUT(CRTCMDOUT) = MCW ;
4 OUTPUT(CRTCMDOUT) = 36H;
5 BAUD$RATE = B2400;
6 OUTPUT(TMCP) = 10110110B; /* COUNTER 2 MODE 3 */
7 OUTPUT(CTR2) = LOW(BAUD$RATE);
8 OUTPUT(CTR2) = HIGH(BAUD$RATE);
9 RETURN;
10 END;

```

147

PL/M-80 COMPILER

PAGE 23

MODULE INFORMATION:

CODE AREA SIZE	= 047DH	1149D
VARIABLE AREA SIZE	= 017CH	380D
MAXIMUM STACK SIZE	= 000AH	10D
870 LINES READ		
0 PROGRAM ERROR(S)		

END OF PL/M-80 COMPILATION


```
    LPMN28 (4) BYTE,  
    PRELL$OUT BYTE PUBLIC, /* PRIORITY INDEX */  
    INACTMOD BYTE, /* ACTUAL USER ID */  
    LPOUTBUF(LPOUTBUFLN) BYTE,  
    /* LP OUTPUT BUFFER */  
    (LPOUTX,LPOUTPTR) BYTE,  
    /* INDICES FOR LP OUTPUT BUFFER  
    LPOUTX - NEXT TO FILL  
    LPOUTPTR - NEXT TO PRINT */  
    LPTELLBACK BYTE,  
    /* TRUE IF TELLBACK MSG REQUESTED AFTER OUTPUT */  
    LPTBCODE BYTE,  
    /* REQUESTED CODE FOR TELLBACK MSG */  
    LPOUTACTIVE BYTE,  
    /* TRUE IF OUTPUT ACTIVE */  
    LPMODSAVE BYTE,  
    /* MODULE NUMBER OF PRINTING MODULE */  
    LPCLEAREND BYTE;
```


LPNEWLINE

POSITION PRINT HEAD AT BEGIN OF NEW LINE

LPNEWLINE: PROCEDURE;

```
IF NOT LPPACK(CR) THEN RETURN;
IF NOT LPPACK(LF) THEN RETURN;
CALL LPSTARTOUT;
RETURN;
END LPNEWLINE;
```

PANEWPAGE

POSITION PRINT HEAD AT TOP OF NEW PAGE

PA\$NEWPAGE: PROCEDURE PUBLIC ;

```
IF NOT LPPACK(FORMFEED) THEN RETURN;  
CALL LPSTARTOUT;  
RETURN;  
END PANEWPAGE;
```

PAPRINT

```
PROCESS PRINT MSG (MN 23)
IF MSGDATA(0) = TRUE : PRINT HEAD AT BEGIN OF NEW LINE
```

```

215 1 PA$PRINT: PROCEDURE PUBLIC ;
216 2 IF MSG(SMN) <> INACTMOD THEN RETURN;
218 2 B3 = MSG(ML) - 5;
219 2 IF B3 = 0 THEN RETURN;
221 2 DO B2 = 1 TO B3;
222 3 /* MOVE CHARACTERS INTO OUTPUT BUFFER */
223 3 /* NOT LPPACK(MSGDATA(B2)) THEN GO TO PR1;
224 3 /* OUTPUT BUFFER FULL */
225 3 END;
226 2 IF MSGDATA(0) THEN CALL LPNEWLINE;
227 2 PR1:
228 2 ;
229 2 CALL LPSTARTOUT;
230 2 RETURN;
231 2 END PAPERINT;

```

```

231 1  $EJECT
232 2  /*
233 2  ****
234 2  ****
235 2  ****
236 2  ****
237 2  ****
238 2  ****
239 2  ****
240 2  ****
241 2  ****
242 2  ****
243 2  ****

```

PAPRINTLINKED

```

231 1  **
232 2  **
233 2  **
234 2  **
235 2  **
236 2  **
237 2  **
238 2  **
239 2  **
240 2  **
241 2  **
242 2  **
243 2  **

```

PROCESS PRINT MSG WITH TELLBACK (MN 25)

```

231 1  /*
232 2  ****
233 2  ****
234 2  ****
235 2  ****
236 2  ****
237 2  ****
238 2  ****
239 2  ****
240 2  ****
241 2  ****
242 2  ****
243 2  ****

```

PAPRINT\$LINKED: PROCEDURE PUBLIC ;

```

231 1  IF MSG(SMN) <> INACTMOD THEN RETURN;
232 2  IF MSG(ML) < 7 THEN RETURN;
233 2  LPTBCODE = MSGDATA(0);
234 2  /* SAVE TELLBACK CODE */
235 2  LPTELLBACK = TRUE;
236 2  ADRMSGDATA = ADRMSGDATA + 1;
237 2  MSG(ML) = MSG(ML) - 1;
238 2  /* ADJUST MSG FOR PROCESS BY LPPRINT */
239 2  LPMODSAVE = MSG(SMN);
240 2  /* SAVE MODULE NUMBER FOR TELLBACK MSG */
241 2  CALL PAPRINT;
242 2  RETURN;
243 2  END PAPRINTLINKED;

```

—*

PA\$OUT\$ACTIVATE

✱

PA\$OUT\$RELEASE

```

257 INACTMOD = 0FFH ;
258 RETURN ;
259 END ;

```

SUBJECT

*/

**

PAIO\$START

INITIALIZE PARALLEL IO MODULE

**

/ *

PAIO\$TART: PROCEDURE PUBLIC;

260 1

CALL CLEARDATA(.LPCLEARBEG,.LPCLEAREND);

261 2

/* CLEAR VARIABLE DATA */

262 2 LPMN26(1)=EX; LPMN26(2)=26; LPMN26(3)=5;

265	2	$\text{LPMN28}(1) = \text{EX}; \text{LPMN28}(2) = 28; \text{LPMN28}(3) = 5;$
-----	---	--

```
268 2 PREL$OUT = ENTERPRIOR(.LPOUTPUT);
```

INACTMOD = 0FFH ;

```

2700 OUTPUT(LPCMDOUT) = LPSTAT; /* MODE 1 OUTPUT */
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000

```

```

271 2 OUTPUT(LPCMDOUT) = UP$STROBE; /* INIT STROBE LEVEL UP */

```

INTMASK 2 272

RETURN;

END PAIO\$START;

274 2

275 1 END;

275 1

4
2
2

MODULE INFORMATION:

CODE	AREA	SIZE	=	Ø1D6H	470D

VARIABLE AREA SIZE = 0213H

MAXIMUM STACK SIZE	SECTION	10D
000AH		

499 LINES READ

Ø PROGRAM ERROR(S)

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE MONITOR
 NO OBJECT MODULE REQUESTED
 COMPILER INVOKED BY: PLM80 :F1:MONI.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)

```

1      MONITOR: DO;
      $NOLIST
      /*
      ***** CONSTANT DATA *****
      */
174 1    DCL EOT LIT '04H'; /* END OF TEXT ID */

175 1    DCL COMMAND$LIST(8) BYTE DATA('C','D','F','G','M','S','T','P'),
      MONITOR$TAG(18) BYTE DATA(1,'MONITOR REQUEST',CPIR+30H),
      MONITOR$IN(11) BYTE DATA(1,'MONITOR IN'),
      MONITOR$OUT(11) BYTE DATA('MONITOR OUT'),
      MON$PROMT(2) BYTE DATA(0,'');
      /*
      ***** VARIABLE DATA *****
      */
176 1    DCL COMMAND BYTE, /* MONITOR COMMAND VALUE */
      (ADR1,ADR2,ADR3) ADDRESS, /* MONITOR PARAMETERS */
      (LOWA,HIGHA) BYTE AT(.ADR1), /* OVERLAP OF ADR1 */
      (BYTE1 BASED ADR1) BYTE,
      SUBS$IN$CHARGE BYTE, /* SUBSTITUTE COMMAND FLAG */
      SAVED$FLAG BYTE,
      SAVED$INSTRUCTION BYTE,
      SAVED$LOCATION ADDRESS, /* DATA FOR MONITOR TRAP */
      END$MON$DATA BYTE;

```

```

$EJECT
/*
*****
: PROMTS CHARACTER .
AND REQUEST AND INPUT MESSAGE STRING FROM
THE SYSTEM CONSOLE.
*****
*/
177 1 RECEIVE$NEXT$COMMAND: PROCEDURE;

178 2 CALL PRINT$SYNC(.MON$PROMT,2,EX);
179 2 CALL CON$INPUT$REQ(EX,TRUE);
180 2 END;

/*
*****
: EXTRACTS COMMAND'S PARAMETERS FROM
MESSAGE INPUT.
*****
*/
181 1 GET$PARAMETERS: PROCEDURE (P1) BYTE;
192 2 DCL P1 BYTE, A ADDRESS,
(ADR BASED A) ADDRESS;

183 2 A = .ADR1 ;
184 2 DO WHILE P1 <> 0 ;
185 3 IF NOT GETNUM THEN DO;
187 4 CALL RECEIVE$NEXT$COMMAND;
188 4 RETURN FALSE;
189 4 END;
190 3 ADR = A4 ;
191 3 A = A + 2 ; P1 = P1 - 1 ;
193 3 END;
194 2 RETURN TRUE ;
195 2 END;

```



```

196 $EJECT
197 /*
198 *****
199 : DUMPS A 16-CHARACTER LINE FROM MEMORY TO
200 THE SYSTEM CONSOLE. ADR1 POINTS TO THE BEGINNING
201 OF THE LINE. ENDS WHEN IT GETS TO ADR2.
202 *****
203 */
204 PRINT$LINE: PROCEDURE PUBLIC;
205 DCL LINE (56) BYTE;
206
207 CALL FILL(.LINE,.LINE+55,' ');/* CLEAR LINE */
208 LINE(0) = 1 ; /* CRLF AFTER PRINT */
209 CALL NUMOUT(ADR1,16,'0',.LINE(1),4);
210 LINE(5) = 'H' ; LINE(6) = ':' ; /* ADDRESS SET */
211 B2 = 8 ;
212 DO WHILE ADR1 <= ADR2 AND B2 < 55 ;
213 CALL CONVASC(BYTE1);
214 LINE(B2) = BU ; LINE(B2+1) = BL ;
215 B2 = B2 + 3 ; ADR1 = ADR1 + 1 ;
216 END;
217 IF ADR1 > ADR2 THEN DO ;
218 CALL PRINT$SYNC(.LINE,56,EX);
219 CALL RECEIVE$NEXT$COMMAND;
220 END;
221 ELSE CALL PRINT$LINKED(.LINE,56,EX,0);
222
223 RETURN;
224 END;

```

```

$EJECT
/*
*****
: DISCONNECTS CURRENT MONITOR FROM THE SYSTEM CONSOLE
  AND REQUEST CONNECTION OF THE INDICATED BY THE
  INPUTED PARAMETER.
*****
*/
CHANGE$COMPUTER: PROCEDURE;
  DCL COMPUTER BYTE;

  IF NOT GET$PARAMETERS(1) THEN RETURN;
  IF LOWA < 1 OR LOWA > 3 THEN CALL RECEIVE$NEXT$COMMAND;
  ELSE DO;
    START(LOWA-1) = LOWA;
    COMPUTER = SHL((LOWA-1),3); /* TIMES 8 */
    CALL CON$RELEASE;
    CALL CON$LINK$REQ(COMPUTER);
  END;

  RETURN;
END;
/*
*****
: FILLS OUT MEMORY BOUNDED BY ADR1 AND ADR2
  WITH CHARACTER INDICATED WITH THIRD PARAMETER.
*****
*/
FILLER: PROCEDURE;
  IF NOT GET$PARAMETERS(3) THEN RETURN;
  B2 = LOW(ADR3);
  CALL FILL(ADR1,ADR2,B2);
  CALL RECEIVE$NEXT$COMMAND;
  RETURN;
END;

```

```

219 1
220 2
221 2
223 2
225 2
226 3
227 3
228 3
229 3
230 3
231 2
232 2

```

```

233 1
234 2
236 2
237 2
238 2
239 2
240 2

```

```

$EJECT
/*
: *****
: : DISPLAYS MEMORY BOUNDED BY ADR1 AND ADR2
:   ON SYSTEM CONSOLE.
: *****
*/
DISPLAY: PROCEDURE;
  IF NOT GET$PARAMETERS(2) THEN RETURN;
  IF ADR1 > ADR2 THEN DO;
    CALL RECEIVE$NEXT$COMMAND;
    RETURN;
  END;

  CALL PRINT$LINE ;
  RETURN;
END;
241 1
242 2
244 2
246 3
247 3
248 3
249 2
250 2
251 2

/*
: *****
: : MOVES MEMORY BLOCK BOUNDED BY ADR1 AND ADR2
:   TO LOCATION POINTED BY ADR3.
: *****
*/
MOVE$MEMORY: PROCEDURE;
  IF NOT GET$PARAMETERS(3) THEN RETURN;
  IF ADR1 > ADR2 OR (ADR2 >= ADR3 AND ADR3 >= ADR1) THEN DO;
    CALL RECEIVE$NEXT$COMMAND;
    RETURN;
  END;

  CALL MOVE(ADR2-ADR1+1,ADR1,ADR3);
  CALL RECEIVE$NEXT$COMMAND;
  RETURN;
END;
252 1
253 2
255 2
257 3
258 3
259 3
260 2
261 2
262 2
263 2

```

```

$EJECT
/*
*****
: GETS A CHARACTER FROM MEMORY AND
: SENDS IT TO THE SYSTEM CONSOLE.
*****
*/
SEND$MEM: PROCEDURE;
  DCL MEMDAT (5) BYTE;

  MEMDAT(0) = 0 ; /* NOT ROLL */
  MEMDAT(1) = ' ';
  CALL CONVASC(BYTE1); /* SET FOR OUTPUT */
  MEMDAT(2) = BU;
  MEMDAT(3) = BL;
  MEMDAT(4) = '-';
  CALL PRINT$SYNC(.MEMDAT,5,EX); /* OUTPUT */
  CALL CON$INPUT$REQ(EX,FALSE);
  RETURN;
END;
/*
*****
: SUBSTITUTE A MEMEORY BYTE
*****
*/
SUBS$NEXT: PROCEDURE;
  B1 = B1 - 1; /* COME BACK TO LAST CHARACTER */
  IF NOT GETNUM THEN DO;
    SUBS$IN$CHARGE = FALSE;
    CALL RECEIVE$NEXT$COMMAND;
    RETURN; END;
  BYTE1 = BL;
  ADRI = ADRI + 1 ;
  CALL SEND$MEM;
  RETURN;
END;

```

```

$EJECT
/*
*****
: TRIGGERS THE SUBSTITUTION SEQUENCE.
  A CHARACTER FROM MEMORY WILL BE DISPLAYED
  AND ANOTHER COMMING FROM THE
  SYSTEM CONSOLE PUT IN ITS PLACE.
*****
*/
SUBSTITUTE: PROCEDURE;
  IF NOT GET$PARAMETERS(1) THEN RETURN;
  SUBS$IN$CHARGE = TRUE;
  CALL SEND$MEM;
  RETURN;
END;
/*
*****
: DISCONNECTS THE MONITOR FROM THE SYSTEM CONSOLE.
  OPTIONALLY, IT PUTS A MONITOR TRAP
  INTO AN INDICATED MEMORY ADDRESS.
*****
*/
GOING: PROCEDURE;
  IF GETNUM THEN DO:
    SAVED$FLAG = TRUE;
    SAVED$LOCATION,ADRI = A4;
    SAVED$INSTRUCTION = BYTE1;
    BYTE1 = 11001111B; /* HST 1 */
    CALL ENTER$INT(1);
  END;
  CALL CON$RFLEAVE;
  CALL PRINT$ASYNC(.MONITOR$OUT,11);
  RETURN;
END;

```

```

289 1
290 2
292 2
293 2
294 2
295 2

```

```

296 1
297 2
299 3
300 3
301 3
302 3
303 3
304 3
305 2
306 2
307 2
308 2

```

```

$EJECT
/*
*****
: TRANSMIT A MESSAGE COMMING FROM THE SYSTEM CONSOLE.
  THE MESSAGE IS SPECIFIED BY A RECEIVER NUMBER
  AND A MESSAGE NUMBER.
*****
*/
309 1 TRANSMIT: PROCEDURE;
310 2   DCL TMSG(4) BYTE;

311 2   IF NOT GET$PARAMETERS(2) THEN RETURN;
313 2   TMSG(1) = EX ; TMSG(3) = 4 ;
315 2   TMSG(0) = LOWA ; TMSG(2) = LOW(ADR2) ;
317 2   IF NOT SEND(.TMSG) THEN DO;
319 3     CALL RECEIVE$NEXT$COMMAND;
320 3     RETURN; END;

322 2   CALL CON$RELEASE;
323 2   CALL PRINT$ASYNCR(.MONITOR$OUT,11);
324 2   RETURN;
325 2   END;
/*****
: CHANGES THE SYSTEM PRINTER TO ANOTHER COMPUTER.
*****
*/
326 1 CHANGE$PRINTER: PROCEDURE;
327 2   DCL PRINTER BYTE;

328 2   IF NOT GET$PARAMETERS(1) THEN RETURN;
330 2   IF LOWA > 0 AND LOWA < 4 THEN /* CORRECT INPUT */
331 2     DO; PRINTER = SHL((LOWA-1),3);
333 3     SYSMN14(0), SYSMN19(0), SYSMN23(0),
334 3     SYSMN24(0), SYSMN25(0) = PRINTER;
335 2     END;
336 2   CALL RECEIVE$NEXT$COMMAND;
  RETURN;

```



```

338      $EJECT
339      /*
          *****
          : USED TO RESTORE ORIGINAL MEMORY VALUES
          WHEN MONITOR TRAP.
          *****
          */
          RESET$INSTRU: PROCEDURE;
                        DCL (P1 BASED SAVED$LOCATION) BYTE;

                        P1 = SAVED$INSTRUCTION;
                        SAVED$FLAG = FALSE;
                        CALL REM$INT(1);
                        RETURN;
                        END;
          /*
          *****
          : EXECUTED BY INTERRUPT HANDLER NUMBER 1, USED FOR
          TRAPPING THE MONITOR IN A MSPECIFIC MEMORY LOCATION.
          *****
          */
          MONITOR$TRAP: PROCEDURE PUBLIC;

                        1
345
                        2
346      IF SAVED$FLAG THEN CALL RESET$INSTRU;
348      CALL MONITOR;
349      RETURN;
350      END;

```

```

351 $EJECT
352 /*****
353 : ALL MONITOR COMMANDS COMMING FROM THE SYSTEM CONSOLE
355 : WILL GET TO THIS PROCEDURE. ONCE THE COMMAND
356 : IS RECOGNIZED THE CONTROL WILL BE CHANGED TO THE
357 : CORRESPONDING PROCEDURE.
359 *****/
360 MONI$INTER: PROCEDURE PUBLIC;
361 B1 = 0 ; /* SET TO BEGINNING OF INPUT STRING */
362 IF NOT GET$CHAR THEN DO;
363 CALL RECEIVE$NEXT$COMMAND;
364 SUBS$IN$CHARGE = FALSE;
365 RETURN; END;
366 IF SUBS$IN$CHARGE THEN DO;
367 CALL SUBS$NEXT;
368 RETURN;
369 END;
370
371 COMMAND = BL;
372 DO B2 = 0 TO LAST(COMMAND$LIST);
373 IF COMMAND$LIST(B2) = COMMAND
374 THEN GOTO INTER$COMMAND; END;
375 CALL RECEIVE$NEXT$COMMAND;
376 RETURN;
377 INTER$COMMAND: ; DO CASE B2;
378 CALL CHANGE$COMPUTER;
379 CALL DISPLAY ;
380 CALL FILLER ;
381 CALL GOING ;
382 CALL MOVE$MEMORY;
383 CALL SUBSTITUTE;
384 CALL TRANSMIT;
385 CALL CHANGE$PRINTER;
386 END;
387 RETURN;
388 END;
389

```

```

$EJECT
/****
*****
: CHECK IF SYSTEM CONSOLE CONNECTION HAS BEEN ACCEPTED,
  AND REQUEST THE FIRST COMMAND FROM THERE.
*****
**/
MONI$INIT: PROCEDURE PUBLIC;
  IF NOT MSG$DATA(0) THEN
    CALL PRINT$ASYNC(.MONITOR$TAG,18);
    ELSE CALL PRINT$ASYNC(.MONITOR$IN,11,EX);
    CALL RECEIVE$NEXT$COMMAND;
  RETURN;
END;

/*
*****
: RECEIVES DISCONNECTION MESSAGE FROM SYSTEM CONSOLE
  AND ENDS THE MONITOR INPUT COMMAND SEQUENCE.
*****
**/
MONI$ENDER: PROCEDURE PUBLIC;
  IF SAVED$FLAG THEN CALL RESET$INSTRU;
  CALL PRINT$ASYNC(.MONITOR$OUT,11);
  RETURN;
END;

397 1  END;

```

MODULE INFORMATION:

CODE AREA SIZE	= 042AH	1066D
VARIABLE AREA SIZE	= 0053H	83D
MAXIMUM STACK SIZE	= 000CH	12D

\$EJECT

/*

SHFT

RETURN BYTE WITH A 1 SHIFTED INTO POSITION P1

INPUT : P1 - BIT POSITION

OUTPUT : P1TH POWER OF 2

**

*/

SHFT: PROCEDURE (P1) BYTE PUBLIC;

DCL P1 BYTE;

IF P1 = 0 THEN RETURN 1;
RETURN ROL(1,P1);
END;42 1
43 2
44 2
45 2
46 2
47 2

\$EJECT

✱

★★

BIT

CHECK A BIT IN A SPECIFIED BYTE

INPUT : P1 - BIT

P2 - BYTE TO CHECK

```

OUTPUT : TRUE IF BIT IS SET, FALSE IF NOT

```

**

—*

BIT: PROCEDURE (P1,P2) BYTE PUBLIC;

DCL

(P1,P2) BYTE;

```
IF (P2 AND SHFT(P1)) = 0 THEN RETURN FALSE;
/* SPECIFIED BIT NOT SET */
```

/* SPECIFIED BIT NOT SET */

RETURN TRUE;

END BIT;

```

$EJECT
/*
*****
**
    CLEAR A BIT IN A SPECIFIED BYTE
    INPUT : P1 - BIT
           P2 - ADDRESS OF BYTE
**
*****
*/
54 1  CLEARBIT:  PROCEDURE (P1,P2) PUBLIC;
55 2  DCL      P1 BYTE,P2 ADDRESS,
           (WORD BASED P2) BYTE;
56 2  WORD = WORD AND (NOT SHFT(P1)) ;
           /* CLEAR BIT */
57 2  RETURN;
58 2  END CLEARBIT;
*****
    CLEARBIT
*****

```

SUBJECT

✱

✱✱

SETBIT

SET A BIT IN A SPECIFIED BYTE

INPUT : P1 - BIT

P2 - ADDRESS OF BYTE

✻

—*

```
59 1 SETEIT: PROCEDURE (P1,P2) PUBLIC;
```

60 2 DCL

P1 BYTE,P2 ADDRESS,
(WORD BASED P2) BYTE;

(WORD BASED P2) BYTE;

```
61 2 WORD = WORD OR SHFT(P1);
```

/* SET BIT */

62 2 RETURN;

```

2
63
END SETBIT;

```

SUBJECT

—*

✱✱

LEGAL

CHECK P1 AGAINST P2

INPUT : P1,P2 VALUES TO CHECK
P3 ERROR CODE FOR SYSMON

PP3 ERROR CODE FOR SYSMON

OUTPUT : TRUE IF P1 < P2

FALSE OTHERWISE

/ *

LEGAL: PROCEDURE (P1,P2,P3) BYTE PUBLIC;

DCL (P1,P2,P3) BYTE;

```
IF P1 < P2 THEN RETURN TRUE;
```

```
CALL ERROR(STACKPTR,0,P3);
```

RETURN FALSE;

END LEGAL;

END SC PUB1;

MODULE INFORMATION:

```

CODE AREA SIZE      = 00D3H      211D
VARIABLE AREA SIZE  = 0015H      21D
MAXIMUM STACK SIZE  = 0006H      6D
301 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-82 COMPILATION

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB2
 NO OBJECT MODULE REQUESTED
 COMPILER INVOKED BY: PIM80 :F1:SC2.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)

```

1      1      SCPUB2:      DO;

      $NOLIST

26      1      LEGAL: PROCEDURE (P1,P2,P3) BYTE EXTERNAL;
27      2      DCL (P1,P2,P3) BYTE;
28      2      END;
29      1      GET: PROCEDURE (P1) EXTERNAL;
30      2      DCL P1 ADDRESS;
31      2      END;
32      1      RELEASE: PROCEDURE (P1) EXTERNAL;
33      2      DCL P1 ADDRESS;
34      2      END;
35      1      ERROR: PROCEDURE(P1,P2,P3) EXTERNAL;
36      2      DCL P1 ADDRESS, (P2,P3) BYTE;
37      2      END;
38      1      VECTOR$ADD: PROCEDURE(P1,P2,P3) EXTERNAL;
39      2      DCL (P1,P2,P3) ADDRESS;
40      2      END;

```


\$EJECT

/*

SETPERTIME

SET NEXT CALL TIME FOR PERIODIC
INPUT: P1 - PERIODIC INDEX

**

*/

48 1 SETPERTIME: PROCEDURE (P1) PUBLIC;

49 2 DCL (P1,CY) BYTE, TIME(4) BYTE;

50 2 XA1 = PERLIST(P1).PERINTADR ;

51 2 B3 = PERINT(4) ;

52 2 XA2 = .PERLIST(P1).PERTIME;

53 2 CALL COPY\$CLOCK(.TIME);
54 2 CALL VECTOR\$ADD(.TIME,XA1,XA2);55 2 RETURN;
56 2 END SETPERTIME;

```

$EJECT
/*
*****
*****
*****
**

```

TIME\$CMP

```

      COMPARE TWO CLOCKS:
      INPUT  : P1 ADDRESS OF FIRST CLOCK VALUE
               P2 ADDRESS OF SECOND CLOCK VALUE
      OUTPUT : TRUE IF (P1) < (P2)
               FALSE OTHERWISE
**
*****
**/
TIME$CMP: PROCEDURE(P1,P2) BYTE PUBLIC;

      DCL
          (P1,P2) ADDRESS,
          (TIM1 BASED P1) (4) BYTE,
          (TIM2 BASED P2) (4) BYTE;

      DO XB3 = 0 TO 3;
          IF TIM1(XB3) < TIM2(XB3) THEN RETURN TRUE;
          IF TIM1(XB3) > TIM2(XB3) THEN RETURN FALSE;
      END;

      RETURN FALSE; /* EQUALS */
      END TIME$CMP;

```

SUBJECT

—*

PERACT

ACTIVATE PERIODIC CALL OF A PROCEDURE

INPUT : P1 - ADDRESS OF PROCEDURE

P2 - ADDRESS OF TIME INTERVAL(RTC FORMAT)

OUTPUT : INDEX OF PERIODIC

PERACT: PROCEDURE (P1,P2) BYTE PUBLIC;

DCCL (P1,P2) ADDRESS, TIME(4) BYTE;

DO XB1 = 0 TO LAST(PERLIST);

SEARCH FOR FREE SLOT */

IF PERLIST(XB1).FREE THEN GO TO PERACT1;

/* FREE SLOT FOUND */

END;

CALL ERROR(.PERACT,0,3);

RETURN OF FH;

PERACT1: PERXTBL(NUMBER) = XB1;

```
PERLIST(XB1).FREE = FALSE;
```

```
PERLIST(XB1).PERADR = P1;
```

```
PERLIST(XB1).PERINTADR = p2;
```

CALL COPY\$CLOCK(.TIME);

```
CALL VECTOR$ADD(.TIME,P2..PERLIST(XB1).PERTIME);
```

/* SET NEXT CALL TIME */

```
NUMBER = NUMBER + 1:
```

```

RETURN PERXTBL(NUMPER-1);

```

END PERACT;

\$EJECT
/*

PERCHG

CHANGE THE TIME INTERVAL OF A PERIODIC CALL
INPUT: P1 - PERIODIC INDEX
P2 - ADDRESS OF NEW TIME INTERVAL

*/

PERCHG: PROCEDURE (P1,P2) PUBLIC;

DCL P1 BYTE,P2 ADDRESS;

IF NOT LEGAL(P1,MAXPER,4) THEN RETURN;
/* ILLEGAL PERIODIC INDEX */

PERLIST(P1).PERINTADR = P2;

/* SET NEW TIME INTERVAL */

CALL SETPERTIME(P1);

/* SET NEW CALL TIME */

RETURN;

END PERCHG;

84 1
85 2
86 2
88 2
89 2
90 2
91 2

SUBJECT

—*

**

PERSUSP

SUSPEND THE PERIODIC CALL OF A PROCEDURE

INPUT: PERIODIC INDEX

✻

一、

PERSUSP: PROCEDURE (P1) PUBLIC;

DCL P1 BYTE;

IF NOT LEGAL(P1,MAXPER,4) THEN RETURN;

/* ILLEGAL PERIODIC INDEX */

```
PERLIST(P1).FREE = TRUE;
```

NUMBER = NUMBER - 1;

IF NUMBER = 0 THEN GO TO SUSP1;

XB2 = FALSE;

DO XB1 = 0 TO NUMBER;

/* REMOVE P1 FROM PERXTBL AND COMPACT PERXTBL */

```
IF NOT XB2 AND PERXTBL(XB1) = P1 THEN XB2 = TRUE;
```

```
IF X#2 THEN PERXTBL(XB1) = PERXTBL(XB1+1);
```

END;

```
SUSP1: PERXTBL(NUMBER) = 0FFH;
```

RETURN;

END;

END SCPUB2;

MODULE INFORMATION:

PL/M-80 COMPILER

PAGE 8

CODE AREA SIZE	= 028DH	653D
VARIABLE AREA SIZE	= 0018H	24D
MAXIMUM STACK SIZE	= 0006H	6D
325 LINES READ		
0 PROGRAM ERROR(S)		

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB3
 NO OBJECT MODULE REQUESTED
 COMPILER INVOKED BY: PLM80 :F1:SC3.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)

1	SCPUB3:	DO;
	\$NOLIST	
26	1	ERROR:PROCEDURE (P1,P2,P3) EXTERNAL;
27	2	DCL P1 ADDRESS,(P2,P3) BYTE;
28	2	END;
29	1	BIT: PROCEDURE (P1,P2) BYTE EXTERNAL;
30	2	DCL (P1,P2) BYTE;
31	2	END;
32	1	PRINT\$ASYNC: PROCEDURE (P1,P2) EXTERNAL;
33	2	DCL P1 ADDRESS, P2 BYTE;
34	2	END;
35	1	CONVASC: PROCEDURE (P1) EXTERNAL;
36	2	DCL P1 BYTE;
37	2	END;

\$EJECT

/*

MSGOVERFLOW

HANDLE MSG OVERFLOW

**

**/

MSGOVERFLOW: PROCEDURE PUBLIC;

38	1	BU = MSG(MN);
39	2	BL = MSG(SMN);
40	2	CODESAVE = A4;
41	2	CALL ERROR(CODESAVE,0,1);
42	2	RETURN;
43	2	END MSGOVERFLOW;
44	2	

\$EJECT
/*

PACKMCB

PACK MCB INTO MSGBUFFER
INPUT: ADDRESS OF MCB

**

**/

PACKMCB: PROCEDURE (A) PUBLIC;

DCL A ADDRESS;

CALL MOVE(4,A,.MSGBUFFER(MSGIN));
MSGIN = MSGIN + 4;

RETURN;
END PACKMCB;

45 1
46 2
47 2
48 2
49 2
50 2


```

61      LASTMSG = MSGIN;
62      MSGIN = 0;
63      END;
64      IF (MSGOUT - MSGIN) < MSG(ML) THEN GO TO SEND2;
        /* NOT ENOUGH SPACE FOR MSG, OVERFLOW */
        SEND1:
66          SAVEMSGIN = MSGIN;
67          CALL PACKMCB(P1);
        /* TRANSFER MCB */
68          MSGIN = SAVEMSGIN + MSG(ML);
        /* COMPUTE BEGIN FOR NEXT MSG */
69          IF MSGIN > LAST(MSGBUFFER) THEN
            /* MSGIN IS OUTSIDE MSGBUFFER */
            DO;
70              LASTMSG = MSGIN;
71              MSGIN = 0;
72              END;
73          NUMMSG = NUMMSG + 1;
74          ADRMSGDATA = .MSGBUFFER(SAVEMSGIN + 4);
75          /* SET ADDRESS OF FIRST DATA BYTE */
76          RETURN TRUE;
        SEND2:
77          ADRMSG = P1;
78          CALL MSGOVERFLOW;
79          RETURN FALSE;
80          END SEND;

```

SUBJECT

$$^*/$$

★★

GET

GET VARIABLE LOCK FOR USER

INPUT: VARIABLE POINTER.

✱
✱

/ *

```
GET: PROCEDURE(KEY) PUBLIC;
```

DCL KEY ADDRESS,

(LOCK BASED KEY) BYTE;

81	1
82	2

SETL1: DO WHILE LOCK <> 0;

END;

LOCK = CPTR;

/* SET LOCK */

IF LOCK <> CPTR THEN GOTO SETL1;

/* ALREADY LOCKED BY OTHER COMPUTER */

RETURN;

END GET;

83	2
84	3
85	2
86	2
88	2
89	2

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
84

—*

* *

RELEASE

RELEASES A VARIABLE LOCK
INPUT: VARIABLE POINTER.

*
*

/ 26

1 RELEASE: PROCEDURE(KEY) PUBLIC;
2 DCL KEY ADDRESS, (LOCK BASED KEY) BYTE;

LOCK = 0;

RETURN;

END RELEASE;

92 93 94

AD-A072 578

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/6 9/2

A TACTICAL SYSTEM EMULATOR FOR A DISTRIBUTED MICRO-COMPUTER ARC--ETC(U)

JUN 79 L A GUILLEN

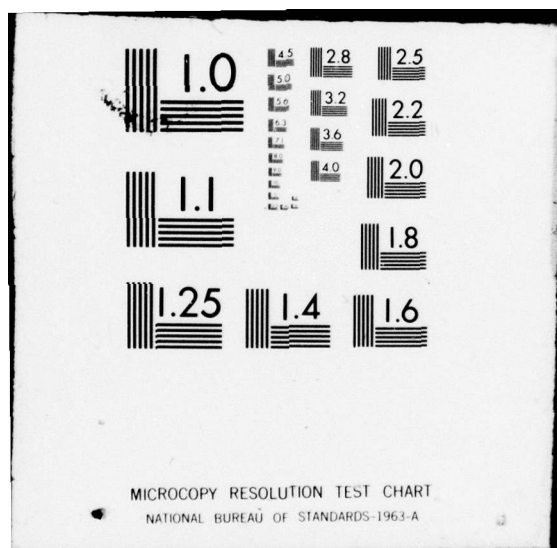
UNCLASSIFIED

NL

3 of 4

AD
A072578





[illegible]

SUBJECT

*/

SENDEXT

```
SEND MSG TO EXTERNAL MODULE
ADDRMSG POINTS TO FIRST BYTE OF MSG
OUTPUT: TRUE IF MSG SENT
        FALSE OTHERWISE
```

✱✱

SENDEXT: PROCEDURE PUBLIC;

CALL GET(.EXTMSGLOCK);

/* SET EXT MSG BUFFER LOCK */

```
IF EXTMSGIN = EXTMSGOUT AND NUMEXTMSG <> 0 THEN GOTO SEXT2;  
/* OVERFLOW */
```

IF EXTMGOUT <= EXTMGIN THEN

DO;

```
IF LAST(EXTMSGBUFFER) - EXTMSGIN >= MSG(ML) THEN GOTO SEXT1;
/* MSG FITS INTO REST OF BUFFER */
```

```
EXTLASTMSG = EXTMMSGIN;
```

```
EXTMSGIN = 0;
```

END;

```
IF EXTMSGOUT - EXTMSGIN < MSG(ML) THEN GOTO SEXT2;  
/* OVERFLOW */
```

SEXT1:

DATE MOVED/Y

CALL MOVE(XB2,ADR\$MSG,.EXTMSGBUFFER(EXTMSGIN));

$$\text{EXTMSGIN} = \text{EXTMSGIN} + \text{XB2} ;$$

/*TRANSFER MSG */

```
IF EXTMSGIN > LAST(EXTMSGBUFFER) THEN
/* EXTMSGIN OUTSIDE BUFFER */
```



```
148 2 DO;  
149 3   EXTLASTMSG = EXTMSGIN;  
150 3   EXTMSGIN = 0;  
151 3   END;  
152 2   NUMEXTMSG = NUMEXTMSG + 1;  
153 2   IF NUMEXTMSG = 1 THEN CALL SETEXTMSG;  
155 2   /* SET RECEIVING CPTR OF NEXT MSG */  
      CALL RELEASE(.EXTMSGLOCK);  
      /* RELEASE(.EXTMSGLOCK) EXT MSG BUFFER */  
      RETURN;  
156 2  
157 2   SECT2: CALL MSGOVERFLOW;  
158 2   CALL RELEASE(.EXTMSGLOCK);  
159 2   RETURN;  
160 2   END SENDTEXT;
```


PL/M-80 COMPILER

PAGE 13

VARIABLE AREA SIZE = 0029H 41D
MAXIMUM STACK SIZE = 0008H 8D
448 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB4
 NO OBJECT MODULE REQUESTED
 COMPILER INVOKED BY: PLM80 :F1:SC4.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)

1	1	SCPUB4:	DO;
		\$NOLIST	
26	1	LEGAL:	PROCEDURE (P1,P2,P3) BYTE EXTERNAL;
27	2		DCL (P1,P2,P3) BYTE;
28	2		END;
29	1	SET\$BIT:	PROCEDURE (P1,P2) EXTERNAL;
30	2		DCL P1 BYTE, P2 ADDRESS;
31	2		END;
32	1	SHFT:	PROCEDURE(P1) BYTE EXTERNAL;
33	2		DCL P1 BYTE;
34	2		END;
35	1	CLEAR\$BIT:	PROCEDURE (P1,P2) EXTERNAL;
36	2		DCL P1 BYTE, P2 ADDRESS;
37	2		END;
38	1	ERROR:	PROCEDURE (P1,P2,P3) EXTERNAL;
39	2		DCL P1 ADDRESS, (P2,P3) BYTE;
40	2		END;

*/

PRIORITY

SCHEDULE THE CALL OF A PRIORITY PROCEDURE
(PRIORITY INT TO BE CALLED FROM INT ROUTINES ONLY)
INPUT: PRIORITY INDEX

PRIORITY: PROCEDURE (P1) PUBLIC;

DCL P1 BYTE;

IF P1 >= MAXPRIOR THEN RETURN;

```

/* ILLEGAL PRIORITY */
CALL SET$BIT(P1,.PRIORSCHEDULE);
RETURN;
END PRIORITY;

```


SUBJECT

*/

[illegible]

ENTERPRIOR

ENTER NEW PRIORITY CALL IN PRIORITY LIST
INPUT : ADDRESS OF PRIORITY PROCEDURE
OUTPUT : PRIORITY INDEX TO PRIORLIST

✱✱

ENTERPRISE: PROCEDURE (P1) BYTE PUBLIC;

DCL P1 ADDRESS;

```
DO XB1 = 0 TO LAST(PRIORLIST);
/* SEARCH FOR FREE SLOT IN PRIORLIST */
```

DO:

PRIORLIST(XB1) = P1;

/* ADDRESS OF PRIORITY PROCEDURE */

RETURN XB1;

END;

END;

```
CALL ERROR(.ENTERPRIOR,0.2);
```

/* PRIORITY LIST OVERFLOW */

RETURN OFFH;

END ENTERPRIOR;

*/

REMPRIOR

INPUT: PRIORITY INDEX

60 1

61 2

201

\$EJECT

*/

**

SETCDC

```

SET CDC INTERRUPT
INPUT : P1 - TIME INTERVAL (LSB = 1.86 MICRO SEC
P2 - ADDRESS TO BE CALLED AT CDC INT
OUTPUT: FALSE IF CDC ALREADY ACTIVE
        TRUE OTHERWISE

```

**

/ *

```

SETCDC: PROCEDURE (P1,P2) BYTE PUBLIC;

```

```
DCL (P1,P2) ADDRESS,
      (P1H,P1L) BYTE AT (.P1);
```

IF CDCACTIVE THEN RETURN FALSE;

CDCACTIVE = TRUE;

DISABLER;

```
INTMASK = INTMASK XOR 1 ;
```

```
OUTPUT(ØDBH) = INTMASK;
```

```
OUTPUT(ØDDH) = P1H;
```

```
OUTPUT(ODDH) = P1L;
```

/* LOAD CTR 1 */

CDCADR = P2;

/* SAVE ADDRESS TO BE CALLED AT CDC INT */

ENABLE;

END; RETURN TRUE;

END SETCDC;

```

82 $EJECT
83 /*
84 ****
85 ****
86 ****
87 ****
88 ****
89 ****
90 **
91
92 ENTER INTERRUPT:P1-PRIORITY LEVEL
93 **
94 ****
95 ****
96 ****
97 ****
98 ****
99 ****
100 **
101
102 ENTER$INT:
103 DCL P1 BYTE;
104
105 INTMASK = INTMASK XOR SHFT(P1);
106 DISABLE;
107 OUTPUT(0DBH) = INTMASK;
108 ENABLE;
109 RETURN;
110 END;

```

*/

REMOVE INTERRUPT: P1 INTERRUPT LEVEL

```
REM$INT: PROCEDURE(P1) PUBLIC;  
DCL P1 BYTE;
```

```
INTMASK = INTMASK OR SHFT(P1);
DISABLE;
OUTPUT(ØDBH) = INTMASK;
ENABLE;
RETURN;
END;
END SCPUB4;
```

90	1
91	2
92	2
93	2
94	2
95	2
96	2
97	2
98	1

```
CODE AREA SIZE          = 0102H      258D
VARIABLE AREA SIZE     = 000AH      10D
MAXIMUM STACK SIZE     = 0004H       4D
308 LINES READ
0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

PRINT\$ASync

PRINT MESSAGE ON CONSOLE
INPUT: P1 - ADDRESS OF TEXT
P2 - NUMBER OF BYTES ON TEXT

205


```
39 1 $EJECT
39 2 /*
40 2 *****
41 2 *****
42 2 *****
44 2 *****
45 2 *****
46 2 *****

PRINT MESSAGE ON CONSOLE
INPUT: P1 - ADDRESS OF TEXT
      P2 - NUMBER OF BYTES ON TEXT
      P3 - SENDER MODULE NUMBER

PRINT$SYNC

*****
*/
PRINT$SYNC: PROCEDURE(P1,P2,P3) PUBLIC;
            DCL P1 ADDRESS,(P2,P3) BYTE;

            SYSMN11(1) = P3;
            SYSMN11(3) = P2 + 4; /* MSG LENGHT */

            IF NOT SEND(.SYSMN11) THEN RETURN;
            CALL MOVE(P2,P1,ADRMSGDATA);

            RETURN;
            END PRINT$SYNC;
```


S E J E C T

✱

★★

CON\$INPUT\$REQ

REQUEST INPUT FROM CONSOLE

INPUT: P1 - SENDER MODULE NUMBER

P2 - ROLL SCREEN FLAG

(IF TRUE ROLL ECREEN AFTER INPUT).

卷之六

* /

CON\$INPUT\$REQ: PROCEDURE(P1,P2) PUBLIC;

DCL (P1,P2) BYTE;

SYSM12(1) = P1 ;/* SENDER NUMBER */

IF NOT SEND(.SYSMN12) THEN RETURN;

MSGDATA(0) = P2;

RETURN;

END CON\$INPUT\$REQ;

47	1	2	2	2	2
48	2				
49					
50					
52					
53					
54					

SUBJECT

—*

✱
✱

CON\$LINK\$REQ

REQUEST LINKAGE WITH CONSOLE
INPUT: P1 - SENDER MODULE NUMBER

* /

CON\$LINK\$REQ: PROCEDURE(P1) PUBLIC;
DCL P1 BYTE;

55	1
56	2

```
SYSMN13(1) = P1; /* SENDER NUMBER */
B1 = SEND(.SYSMN13);
```

57 22
58 22

RETURN;

END CON\$LINK\$REQ;

59 2
60 2

*/

PAR\$LINK\$REQ

REQUEST LINKAGE WITH PRINTER
INPUT: P1 - SFENDER MODULE NUMBER

*/

PAR\$LINK\$REQ: PROCEDURE(P1) PUBLIC;
DCL P1 BYTE;

61	1
62	2

```
SYSMN14(1) = P1 ;/* SENDEF NUMBER */
B1 = SEND(.SYSMN14);
```

63 2
64 2

RETURN;

END;

65	2
66	2

209

*/

BEGIN NEW TEXT ON CONSOLE
INPUT: P1 - SENDER NUMBER

```
CON$BEG$LINE: PROCEDURE(P1) PUBLIC;  
              DCL P1 BYTE;
```

```

SYSMN17(1) = P1;
IF NOT SEND(.SYSMN17) THEN RETURN;
RETURN;
END CON$BEG$LINE;

```

77	1	2	2	2	2
78					
79					
80					
82					
83					

212


```

$EJECT
/*
*****
**
**
WRITE IN SYSTEM PRINTER
**
*****
**/
WRITE: PROCEDURE (P1,P2,P3) PUBLIC;
      DCL P1 ADDRESS, (P2,P3) BYTE;

      SYSMN23(1) = P3; SYSMN23(3) = P2 + 4 ;
      IF NOT SEND(.SYSMN23) THEN RETURN;
      CALL MOVE(P2,P1,ADRMMSGDATA);
      RETURN;
      END;
105 1
106 2
107 2
109 2
111 2
112 2
113 2

```

WRITE

WRITE IN SYSTEM PRINTER

SUBJECT

✱

*
*

WRITELINKED

WRITE IN SYSTEM PRINTER
AND WAIT FOR AN ANSWER

※

* /

WRITE\$LINKED: PROCEDURE (P1,P2,P3,P4) PUBLIC;
DCL P1 ADDRESS, (P2,P3,P4) BYTE;

114	1
115	2

```

1116 2      SYSMN25(1)=P3; SYSMN25(3)=P2 + 5 ;
1118 2      IF NOT SEND(.SYSMN25) THEN RETURN;
1200 2      MSGDATA(Ø) = P4 ; /* TELLBACK CODE */
121 2      CALL MOVE(P2,P1,ADMSGDATA+1);
122 2      RETURN;
123 2      END;

```

124 1 END SCPUB5;

MODULE INFORMATION:

```
CODE AREA SIZE      = 01E3H      483D
VARIABLE AREA SIZE  = 001CH      28D
MAXIMUM STACK SIZE  = 0004H      4D
397 LINES READ
0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```

1      SCPU6: DO;
28    $NOLIST
1      BIT: PROCEDURE (P1,P2) BYTE EXTERNAL;
27    DCL (P1,P2) BYTE;
28    END;
/*
***
**
CHECK ACTIVE STATUS OF MODULE
INPUT: MODULE NUMBER
OUTPUT: TRUE IF MODULE ACTIVATED,
        FALSE OTHERWISE.
**
****
*/
ACTIVE
*****
1      ACTIVE: PROCEDURE (P1) BYTE PUBLIC;
27    DCL P1 BYTE;
30    RETURN BIT(1,MODSTATUS(P1));
31    END ACTIVE;
32

```

\$/**
\$EJECT

★

✱

UPDSTAT

UPDATE MODULE STATUS IN MODSTATUS TABLE

INPUT: P1 - MODULE NUMBER

P2 - STATUS

✱

—
24

UPDSTAT: PROCEDURE (P1,P2) PUBLIC;

DCL

(P1, P2) BYTE;

MODSTATUS(P1) = P2;

RETURN;

END UPDSTAT;

33 1

34 2

35 2

36 2

37 2

*/

MOVE POINTER B1 TO 1ST NOM-BLANK IN MSGDATA
STARTING AT MSGDATA(B1)

```

48 1 DEBLK: PROCEDURE PUBLIC;
49 2 DO WHILE MSGDATA(B1) = ' ' AND B1 <= MSG(ML) - 5;
50 3 B1 = B1 + 1;
51 3 END;
52 2 RETURN;
53 2 END DEBLK;

```



```

54 $EJECT
55 /*
56 ****
57 ****
58 ****
59 ****
60 ****
61 ****
62 ****
63 ****
64 ****
65 ****
66 ****
67 ****
68 ****
69 ****
70 ****
71 ****
72 ****
73 ****

        GETNUM

        CONVERT ASCII CODES IN MSGDATA TO HEX NUMBER
        STARTIN AT MSGDATA(B1)
        CONVERTED NUMBER IS LEFT IN A4
        RETURN TRUE IF NUMBER IS O.K.,FALSE OTHERWISE

        **
        ****
        */
        GETNUM: PROCEDURE BYTE PUBLIC;
        DCL EOT LIT '04H'; /* END OF TASK ID */
        DCL (BASE10,BASE16) ADDRESS,
            MORE$STRING BYTE,
            DIGITS(*) BYTE DATA('0123456789ABCDEF');

        CALL DEELK; /* SKEEP BLANKS */
        IF MSGDATA(B1) = EOT THEN RETURN FALSE;

        BASE10,BASE16 = 0; MORE$STRING = TRUE;
        DO WHILE MORE$STRING;
            MORE$STRING = FALSE;
            XB1 = MSGDATA(B1);
            DO XB2 = 0 TO LAST(DIGITS);
                IF XB1 = DIGITS(XB2) THEN
                    DO;
                        BASE10 = SHL(BASE10,3) + SHL(BASE10,1) + XB2;
                        BASE16 = SHL(BASE16,4) + XB2;
                        B1 = B1 + 1;
                    END;
                    MORE$STRING = TRUE;
                END;
            END;
        END;

```

74	3	END;	
75	2	IF XB1 = 'H' THEN DO;	
77	3	A4 = BASE16 ; B1 = B1 + 1 ;	
79	3	RETURN TRUE;	
80	3	END;	
81	2	IF XB1 = ' ' OR XB1 = '-' OR XB1 = ',' OR	
		XB1 = '.' THEN DO;	
83	3	A4 = BASE10 ; B1 = B1 + 1 ;	
85	3	RETURN TRUE;	
86	3	END;	
87	2	IF XB1 = EOT THEN DO;	
89	3	A4 = BASE10;	
90	3	RETURN TRUE;	
91	3	END;	
92	2	RETURN FALSE;	
93	2	END;	

```

94 1  $EJECT
95 1  /*
    2  ****
    3  ****
    4  ****
    5  ****
    6  ****
    7  ****
    8  ****
    9  ****
   10  ****
   11  ****
   12  ****
   13  ****
   14  ****
   15  ****
   16  ****
   17  ****
   18  ****
   19  ****
   20  ****
   21  ****
   22  ****
   23  ****
   24  ****
   25  ****
   26  ****
   27  ****
   28  ****
   29  ****
   30  ****
   31  ****
   32  ****
   33  ****
   34  ****
   35  ****
   36  ****
   37  ****
   38  ****
   39  ****
   40  ****
   41  ****
   42  ****
   43  ****
   44  ****
   45  ****
   46  ****
   47  ****
   48  ****
   49  ****
   50  ****
   51  ****
   52  ****
   53  ****
   54  ****
   55  ****
   56  ****
   57  ****
   58  ****
   59  ****
   60  ****
   61  ****
   62  ****
   63  ****
   64  ****
   65  ****
   66  ****
   67  ****
   68  ****
   69  ****
   70  ****
   71  ****
   72  ****
   73  ****
   74  ****
   75  ****
   76  ****
   77  ****
   78  ****
   79  ****
   80  ****
   81  ****
   82  ****
   83  ****
   84  ****
   85  ****
   86  ****
   87  ****
   88  ****
   89  ****
   90  ****
   91  ****
   92  ****
   93  ****
   94  ****
   95  ****
   96  ****
   97  ****
   98  ****
   99  ****
  100  ****
  101  ****
  102  ****
  103  ****
  104  ****
  105  ****
  106  ****
  107  ****
  108  ****

221  ADD TWO 4 BYTES VECTORS
    2  **
    3  **
    4  **
    5  **
    6  **
    7  **
    8  **
    9  **
   10  **
   11  **
   12  **
   13  **
   14  **
   15  **
   16  **
   17  **
   18  **
   19  **
   20  **
   21  **
   22  **
   23  **
   24  **
   25  **
   26  **
   27  **
   28  **
   29  **
   30  **
   31  **
   32  **
   33  **
   34  **
   35  **
   36  **
   37  **
   38  **
   39  **
   40  **
   41  **
   42  **
   43  **
   44  **
   45  **
   46  **
   47  **
   48  **
   49  **
   50  **
   51  **
   52  **
   53  **
   54  **
   55  **
   56  **
   57  **
   58  **
   59  **
   60  **
   61  **
   62  **
   63  **
   64  **
   65  **
   66  **
   67  **
   68  **
   69  **
   70  **
   71  **
   72  **
   73  **
   74  **
   75  **
   76  **
   77  **
   78  **
   79  **
   80  **
   81  **
   82  **
   83  **
   84  **
   85  **
   86  **
   87  **
   88  **
   89  **
   90  **
   91  **
   92  **
   93  **
   94  **
   95  **
   96  **
   97  **
   98  **
   99  **
  100  **
  101  **
  102  **
  103  **
  104  **
  105  **
  106  **
  107  **
  108  **

    2  VECTOR$ADD: PROCEDURE (P1,P2,P3) PUBLIC;
    3  1  DCL (P1,P2,P3) ADDRESS,
    4  2  (ELE1 BASED P1) BYTE,
    5  3  (ELE2 BASED P2) BYTE,
    6  4  (ELE3 BASED P3) BYTE,
    7  5  CY BYTE;
    8  6
    9  7
   10 8
   11 9
   12 A
   13 B
   14 C
   15 D
   16 E
   17 F
   18 G
   19 H
   20 I
   21 J
   22 K
   23 L
   24 M
   25 N
   26 O
   27 P
   28 Q
   29 R
   30 S
   31 T
   32 U
   33 V
   34 W
   35 X
   36 Y
   37 Z
   38
   39
   40
   41
   42
   43
   44
   45
   46
   47
   48
   49
   50
   51
   52
   53
   54
   55
   56
   57
   58
   59
   60
   61
   62
   63
   64
   65
   66
   67
   68
   69
   70
   71
   72
   73
   74
   75
   76
   77
   78
   79
   80
   81
   82
   83
   84
   85
   86
   87
   88
   89
   90
   91
   92
   93
   94
   95
   96
   97
   98
   99
  100
  101
  102
  103
  104
  105
  106
  107
  108

    2  P1 = P1 + 3;
    3  P2 = P2 + 3;
    4  P3 = P3 + 3;
    5  CY = 0 ;
    6  DO XB1 = 0 TO 3;
    7  1  ELE3 = ELE1 + ELE2 + CY;
    8  2  CY = 0 PLUS 0 ;
    9  3  P1=P1-1; P2=P2-1; P3=P3-1;
   10 4  END;
   11 5  RETURN;
   12 6  END;
   13 7
   14 8
   15 9
   16 A
   17 B
   18 C
   19 D
   20 E
   21 F
   22 G
   23 H
   24 I
   25 J
   26 K
   27 L
   28 M
   29 N
   30 O
   31 P
   32 Q
   33 R
   34 S
   35 T
   36 U
   37 V
   38 W
   39 X
   40 Y
   41 Z
   42
   43
   44
   45
   46
   47
   48
   49
   50
   51
   52
   53
   54
   55
   56
   57
   58
   59
   60
   61
   62
   63
   64
   65
   66
   67
   68
   69
   70
   71
   72
   73
   74
   75
   76
   77
   78
   79
   80
   81
   82
   83
   84
   85
   86
   87
   88
   89
   90
   91
   92
   93
   94
   95
   96
   97
   98
   99
  100
  101
  102
  103
  104
  105
  106
  107
  108

```


[illegible]

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB7
 NO OBJECT MODULE REQUESTED
 COMPILER INVOKED BY: PLM80 :F1:SC7.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)

1	SCPub7:	DO;
	\$NOLIST	
26	NUMOUT:	PROCEDURE (P1,P2,P3,P4,P5) EXTERNAL;
27		DCL (P1,P4) ADDRESS, (P2,P3,P5) BYTE;
28		END;
29	PRINT\$ASYNC:	PROCEDURE (P1,P2) EXTERNAL;
30		DCL P1 ADDRESS, P2 BYTE;
31		END;
32	CON\$LINK\$REQ:	PROCEDURE (P1) EXTERNAL;
33		DCL P1 BYTE;
34		END;
35	UPDSTAT:	PROCEDURE (P1,P2) EXTERNAL;
36		DCL (P1,P2) BYTE;
37		END;

—*

ERROR

SYSTEM ERROR
MAY BE CALLED WHEN AN INTERNAL ERROR CONDITION OCCURS,
PROGRAM LIMITS ARE EXCEEDED ETC.
ERROR GENERATES AN ASYNCHRONOUS PRINTOUT TO INDICATE
THE NATURE AND LOCATION OF ERROR CONDITION .
INPUT: P1 - LOCATION OF ERROR OR OTHER ADDRESS VALUE
P2, P3 - BYTE VALUES TO SPECIFY ERROR

—*

```

50      1  ERROR :      PROCEDURE ( P1,P2,P3) PUBLIC;
51      2  DFCLARE      P1 ADDRESS, (P2,P3) BYTE,
52      2              (STACKTOP BASED P1) ADDRESS;
53      2              CALL NUMOUT(STACKTOP,16,'',ERRORMSG(7),4);
54      2              CALL NUMOUT(DOUBLE(P2),10,'',ERRORMSG(11),3);
55      2              CALL NUMOUT(DOUBLE(P3),10,'',ERRORMSG(14),3);
56      2              CALL PRINT$ASYNC(.ERRORMSG,17);
57      2              /* SEND ASYNC PRINT MSG TO CRT */
58      2              RETURN;
59      2              END ERROR;
60      1  END SCPUB7;

```

MODULE INFORMATION:

CODE	AREA	SIZE	
	= 0098H		152D

VARIABLE AREA SIZE = 0007H	7D
MAXIMUM STACK SIZE = 0008H	8D
223 LINES READ	
0 PROGRAM ERROR(S)	

END OF PL/M-80 COMPILATION

EMULATOR PROGRAM LISTING

The following table summarizes the contents of emulator files.

No.	File name	Procedure	Page
1	emula1	emula1	231
		send\$msgs	232
		emulate	233
		correct\$input\$msg	234
		correct\$id	235
		periodic\$mod	236
		demand\$mod	237
		data\$filler	238
		set\$mods	239
2	emula2	emula2	241
		enter\$mod	252
		clear\$mod	252
		get\$task	253
		print\$time	254
		set\$out\$msg	255
		subs\$data	257
		move\$task	259
		emu\$count	261
		emu\$counter	261
		start\$emu	262
		send\$write\$msgs	263
		write\$data	265
		disconnect\$crt	265
		check\$se\$connection	266
		init\$emu	266
		stop\$emu	267
		msgent3	268
		emula2.main	269
3	emula3	emula3	270
		wrt	273
		wrt\$link	273
		set\$buffers	274
		prn\$data\$per	275
		prn\$data\$dem	276
		prn\$order	278
		prn\$head\$per	279
		prn\$head\$dem	280
		prn\$per	281
		prn\$dem	281
		stt\$head	283
		stt\$head\$per	283
		stt\$head\$dem	284
		stt\$head\$msg	284
		stt\$data	285

	stt\$data\$msg	286
	stt\$msg	287
	prn\$stt\$per	288
	prn\$stt\$dem	289
	prn\$stt\$msg	290
	reset\$writer	290
	prn	291
	rec\$costbcode	292
	send\$task	293
	rec\$task	295
	set\$new\$sink	296
	set\$writer	297
	msgent4	298
4	emula4	
	emula4	300
	get\$number	304
	send\$info	304
	quest	305
	save\$task	305
	save\$delay	306
	check\$msg\$id	306
	check\$task\$id	307
	prn\$sequence	307
	check\$less\$than	308
	slave1	308
	slave2	309
	state00-state32	310
	receive\$input	320
	write\$ack\$received	322


```

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EMULA1
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY: PLMB0 :F1:EMULA1.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)

```

```

*****
EMULAL1: *****
/******
*****
THIS MODULE CONTAINS THE PERIODIC AND DEMAND EMULATOR PROCEDURES
,THE NETWORK INFORMATION DATA RECEIVER AND THE ROUTINE TO INITIA-
TE THEM INTO THE SYSTEM.
*****
*/
*****
```

DO;			
\$NOLIST			
99	1	DCL DBD LIT	'DATA\$BLOCK\$DEM';
		DBP LIT	'DATA\$BLOCK\$PER';
		DTA LIT	'MSG\$DATA';
		MBK LIT	'MOD\$BLOCK';

```

100      $EJECT
101      /*
102      *****
103      SEND MESSAGES FROM SOURCE TO SINK TASK
104      INPUT: POINTER TO ELEMENT NUM$MSG$OUT IN ACTIVATION RECORD.
105      *****
106      */
107      SEND$MSGS: PROCEDURE (ADR$ELEMENT);
108      DCL NUM BYTE, ADR$ELEMENT ADDRESS;
109      DCL (MSG$ELEMENT BASED ADR$ELEMENT) BYTE;
110
111      MSG$HEADER (SMN) = CPTR$ID + 1 ;
112      NUM = MSG$ELEMENT;
113      IF NUM = 0 THEN RETURN;
114
115      CALL COPY$CLOCK(.MSG$TEXT(1)); /* GET SENDING TIME */
116
117      DO B1 = 1 TO NUM; /* SEND ALL MSGS */
118      ADR$ELEMENT = ADR$ELEMENT + 1 ;
119      MSG$HEADER (RMN) = MSG$ELEMENT;
120      ADR$ELEMENT = ADR$ELEMENT + 1 ;
121      MSG$HEADER (MN) = MSG$ELEMENT;
122      ADR$ELEMENT = ADR$ELEMENT + 1 ;
123      MSG$TEXT (0) = MSG$ELEMENT;
124      ADR$ELEMENT = ADR$ELEMENT + 1 ;
125      MSG$HEADER (ML) = MSG$ELEMENT + 9 ;
126      IF NOT SEND(.MSG$HEADER) THEN CALL ILLEGALMSG;
127      ELSE CALL MOVE(5,.MSG$TEXT,ADRM$GDATA);
128      END;
129      RETURN;
130      END SEND$MSGS;

```

```

123      $EJECT
124      /*
125      ****
126      ****
127      ****
128      ****
129      ****
130      ****
131      ****
132      ****
133      ****
134      ****
135      ****
136      ****

EMULATE: EMULATE DELAY TIME AND GET CONSUMED TIME
INPUT: POINTER TO ELEMENT DELAY IN ACTIVATION RECORD.
****
*/
EMULATE: PROCEDURE (DAT$ADR);
    DCL DAT$ADR ADDRESS;
    DCL (DAT BASED DAT$ADR) BYTE;

    DO B1 = 1 TO DAT; /* NUMBER OF MILLISECONDS */
        CALL TIME (10);
    END;

    DAT$ADR = DAT$ADR + 1 ;
    DAT = DAT + 1 ; /* INCREMENT NUMBER OF EXECUTIONS */
    DAT$ADR = DAT$ADR + 1;
    CALL COPY$CLOCK(.TEMP$CLOCK2); /* GET ACTUAL TIME */
    CALL VECTOR$SUB(.TEMP$CLOCK2,.TEMP$CLOCK1,.TEMP$CLOCK3);
    CALL VECTOR$ADD(.TEMP$CLOCK3,DAT$ADR,DAT$ADR);
    RETURN;
END EMULATE;

```

```

137 $EJECT
138 /*
139 *****
140 *****
141 *****
142 *****
143 *****
144 *****
145 *****
146 *****
147 *****
148 *****
149 *****
150 *****
151 *****
152 *****
153 *****
154 *****
155 *****
156 *****
157 *****

CORRECT$INPUT$MSG: VERIFIES WHETHER THE MSG JUST ARRIVED IS O.K.
INPUT: MSG$ID - ARRIVED MESSAGE NUMBER.
      ADR$ELEMENT - POINTER TO ELEMENT NUM$MSG$IN
      IN ACTIVATION RECORD.
*****
*/
CORRECT$INPUT$MSG: PROCEDURE (MSG$ID,ADR$ELEMENT) BYTE;
      DCL (MSG$ID,NUM$MSG) BYTE, ADR$ELEMENT ADDRESS;
      DCL (ELEMENT BASED ADR$ELEMENT) BYTE;

      NUM$MSG = ELEMENT;
      IF NUM$MSG = 0 THEN RETURN FALSE;
      ADR$ELEMENT = ADR$ELEMENT + 1;
      DO B1 = 1 TO NUM$MSG; /* CHECK AMOUNT MSGS */
      IF MSG$ID = ELEMENT THEN DO;
        ADR$ELEMENT = ADR$ELEMENT + 1;
        ELEMENT = ELEMENT + 1; /* INCRE # RECEIVED */
        ADR$ELEMENT = ADR$ELEMENT + 1;
        CALL VECTOR$SUB(.TEMP$CLOCK1,.DTA(1),.TEMP$CLOCK2);
        CALL VECTOR$ADD(.TEMP$CLOCK2,ADR$ELEMENT,ADR$ELEMENT);
        RETURN TRUE;
      END;
      ELSE ADR$ELEMENT = ADR$ELEMENT + 6; /* POINT TO NEXT */
      END;
      RETURN FALSE; /* INVALID INPUT MSGS */
      END CORRECT$INPUT$MSG;

```

```

158 $EJECT
159 /*
160 ****
161 ****
162 ****
163 ****
164 ****
165 ****
166 ****
167 ****
168 ****
169 ****
170 ****

CORRECT$ID: CHECK IF TASK DEMANDING TASK IS CORRECT
INPUT: ID - DEMANDING TASK NUMBER.
      MAX - MAXIMUM OF TASKS FOR SERVICE.
      ADR$ELEMENT - POINTER TO ELEMENT NUMBER IN ACTIVATION RECORD.
      STEP - ACTIVATION RECORD SIZE.
****
*/
CORRECT$ID: PROCEDURE (ID, MAX, ADR$ELEMENT, STEP) BYTE;
DCL (ID, MAX) BYTE, (ADR$ELEMENT, STEP) ADDRESS;
DCL (ELEMENT BASED ADR$ELEMENT) BYTE;

DO B1 = 1 TO MAX; /* LOOK AMOUNT ALL ALLOWED */
  IF ID = ELEMENT THEN DO;
    MBK = B1 - 1; /* SET POINTER */
    RETURN TRUE;
  END;
  ELSE ADR$ELEMENT = ADR$ELEMENT + STEP; /* POINT TO NEXT */
  END;
  RETURN FALSE; /* INVALID TASK ID */
END CORRECT$ID;

```



```

171 1 171 1 $EJECT
172 2 172 2 /*
174 3 174 3 *****
175 3 175 3 *****
176 3 176 3 *****
177 2 177 2 *****
178 2 178 2 *****
179 2 179 2 *****
180 2 180 2 *****
181 2 181 2 *****

PERIODIC$MOD: TAKES CARE OF ANY PERIODIC TASK SCHEDULED
*****
*/
PERIODIC$MOD: PROCEDURE;

171 1 171 1 IF NOT CORRECT$ID(B3.NUM$PER$TASK,.DBP(0).NUMBER,
172 2 172 2 PER$BLOCK$LEN) THEN DO;
174 3 174 3 CALL ERROR(.PERIODIC$MOD,1,1);
175 3 175 3 RETURN;
176 3 176 3 END;
177 2 177 2 CALL COPY$CLOCK(.TEMP$CLOCK1); /* SAVE ENTRY TIME */
178 2 178 2 CALL EMULATE(.DBP(MBK).DELAY);
179 2 179 2 CALL SEND$MSG(.DBP(MBK).NUM$MSG$OUT);
180 2 180 2 RETURN;
181 2 181 2 END PERIODIC$MOD;

```


237

```

205 1  $EJECT
206 2  /*
208 2  *****
210 3  *****
211 3  *****
212 3  *****
213 2  *****
214 3  *****
215 3  *****
216 3  *****
217 3  *****
218 3  *****
219 3  *****
220 3  *****
221 3  *****
222 3  *****
223 3  *****
224 3  *****
225 3  *****
226 3  *****
227 3  *****
228 3  *****
229 4  *****
230 4  *****
231 3  *****

DATA$FILLER: RECEIVES INFORMATION ABOUT NETWORK DATA
*/

DATA$FILLER: PROCEDURE;

  IF MSG(MN) = 0 THEN RETURN; /* NO USE */

  IF MSG(MN) > 17 THEN DO;
    CALL ILLEGALMSG;
    RETURN;
  END;

  DO CASE MSG(MN);

    RETURN; /* MN 0 */
    NUM$DEM$TASK = DTA(0); /* MN 1 */
    DBP(DTA(0)).NUMBER = DTA(1); /* MN 2 */
    DBP(DTA(0)).DELAY = DTA(1); /* MN 3 */
    CALL MOVE(2,.DTA(1),.DBP(DTA(0)).PERIOD(2));
    DBP(DTA(0)).NUM$MSG$OUT = DTA(1); /* MN 5 */
    CALL MOVE(4,.DTA(2),.DBP(DTA(0)).MSG$DATA(DTA(1)));
    DBD(DTA(0)).NUMBER = DTA(1); /* MN 7 */
    DBD(DTA(0)).DELAY = DTA(1); /* MN 8 */
    DBD(DTA(0)).NUM$OF$STAT,DBD(DTA(0)).STAT$NUM=DTA(1);
    DBD(DTA(0)).NUM$MSG$OUT = DTA(1); /* MN 10 */
    CALL MOVE(4,.DTA(2),.DBD(DTA(0)).MSG$DATA(DTA(1)));
    DBD(DTA(0)).NUM$MSG$IN = DTA(1); /* MN 12 */
    DBD(DTA(0)).MSG$IN$DATA(DTA(1)) = DTA(2); /* MN 13 */
    DO B1 = 1 TO NUM$PER$TASK; /* MN 14 */
      DBP(B1-1).INDEX = PERACT(.PERIODIC$MOD,.DBP(B1-1).PERIOD);
    END;
    CALL CLEAR$DATA(.NUM$PER$TASK,.MOD$BLOCK);

```


PL/M-80 COMPILER

PAGE 10

CODE AREA SIZE = 0650H 1616D
VARIABLE AREA SIZE = 000FH 15D
MAXIMUM STACK SIZE = 0006H 6D
511 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION

1 EMULA2: D0;

THIS PROGRAM CONTAINS PROCEDURES FOR
CONTROLLING THE EMULATION SEQUENCE.

\$ INCLUDE (:F1:EMUDTP.SRC)

241

UPDATE: 4

SYSTEM DATA DECLARATIONS

```
DCL TRUE LIT '0FFH',
      FALSE LIT '0',
      RMN LIT '0',
      SMN LIT '1',
      MN LIT '2'.
```



```

=1 ML LIT '3',
=1 MAXCPTL LIT '3',
=1 MAXMOD LIT '8',
=1 MAXSYSMOD LIT '24',
=1 MSGBUFLN LIT '512',
=1 MAXPRIOR LIT '8',
=1 MAXPER LIT '8',
=1 BEGINCM LIT '0F400H',
=1 INTVECTOR LIT '03000H',
=1 CPTL LIT '1',
=1 FIRSTMN LIT '0',
=1 LASTMN LIT 'FIRSTMN+7',
=1 EX LIT 'FIRSTMN',
=1 MSG$TBL$ADR LIT '0F660H';
/* MAX NUMBER OF COMPUTERS */
/* MAX NUMBER OF MODULES IN 1 CPTL */
/* MAX NUMBER OF MODULES IN SYSTEM */
/* LENGTH OF MSG BUFFER */
/* MAX NUMBER OF PRIORITY CALLS */
/* MAX NUMBER OF PERIODIC CALLS */
/* BEGIN OF COMMON MEMORY */
/* INTERRUPT VECTOR */
/* NUMBER OF HOST COMPUTER */
/* MN OF FIRST MODULE IN CPTL */
/* MN OF LAST MODULE IN CPTL */
/* MN OF EXEC MODULE */
/* MSG. ENTRY TABLE ADDRESS */

/*
***
*/

4 1 DCL (B1,B2,B3) BYTE EXTERNAL,
      (A1,A2,A3,A4) ADDRESS EXTERNAL,
      (BL,BU) BYTE AT (.A4);
/*
SYSTEM WORK VARIABLES
BU(UPPER) AND BL(LOWER) OVERLAY ADDRESS VARIABLE A4
*/

5 1 DCL (ADRMMSG,ADRMMSGDATA) ADDRESS EXTERNAL,
      (MSG BASED ADRMSG) (4) BYTE,
      (MSGDATA BASED ADRMSGDATA) (100) BYTE;
/*
WORK AREAS FOR MSG CONTROL BLOCK (MSG) AND MSG
DATA BYTES (MSG DATA)
ADRMMSG AND ADRMSGDATA ARE SET BY EXEC BEFORE THE
CALL OF A MODULES MSG HANDLER
*/

6 1 DCL INTMASK BYTE EXTERNAL;

```


243

```

25 2 =1      END;
26 1 =1      ILLEGALMSG: PROCEDURE EXTERNAL;
27 2 =1      END;
28 1 =1      PRIORITY: PROCEDURE (P1) EXTERNAL;
29 2 =1      DCL P1 BYTE;
30 2 =1      END;
31 1 =1      ENTER$PRIOR: PROCEDURE (P1) BYTE EXTERNAL;
32 2 =1      DCL P1 ADDRESS;
33 2 =1      END;
34 1 =1      REMPRIOR: PROCEDURE (P1) EXTERNAL;
35 2 =1      DCL P1 BYTE ;
36 2 =1      END;
37 1 =1      SET$CDC: PROCEDURE (P1,P2) BYTE EXTERNAL;
38 2 =1      DCL (P1,P2) ADDRESS;
39 2 =1      END;
40 1 =1      PRINT$ASYNC: PROCEDURE (P1,P2) EXTERNAL;
41 2 =1      DCL P1 ADDRESS, P2 BYTE;
42 2 =1      END;
43 1 =1      PRINT$SYNC: PROCEDURE (P1,P2,P3) EXTERNAL;
44 2 =1      DCL P1 ADDRESS, (P2,P3) BYTE;
45 2 =1      END;
46 1 =1      CON$INPUT$REQ: PROCEDURE (P1,P2) EXTERNAL;
47 2 =1      DCL (P1,P2) BYTE;
48 2 =1      END;
49 1 =1      CON$LINK$REQ: PROCEDURE (P1) EXTERNAL;
50 2 =1      DCL P1 BYTE;
51 2 =1      END;

```

```

52 1 1 PAR$LINK$REQ: PROCEDURE (P1) EXTERNAL;
53 2 1 DCL P1 BYTE;
54 2 1 END;
55 1 1 CON$RELEASE: PROCEDURE EXTERNAL;
56 2 1 END;
57 1 1 PRINT$LINKED: PROCEDURE (P1,P2,P3,P4) EXTERNAL;
58 2 1 DCL P1 ADDRESS, (P2,P3,P4) BYTE;
59 2 1 END;
60 1 1 PAR$RELEASE: PROCEDURE EXTERNAL;
61 2 1 END;
62 1 1 PAGE: PROCEDURE (P1) EXTERNAL;
63 2 1 DCL P1 BYTE;
64 2 1 END;
65 1 1 WRITE: PROCEDURE (P1,P2,P3) EXTERNAL;
66 2 1 DCL P1 ADDRESS, (P2,P3) BYTE;
67 2 1 END;
68 1 1 WRITE$LINKED: PROCEDURE (P1,P2,P3,P4) EXTERNAL;
69 2 1 DCL P1 ADDRESS, (P2,P3,P4) BYTE;
70 2 1 END;
71 1 1 UPDSTAT: PROCEDURE (P1,P2) EXTERNAL;
72 2 1 DCL (P1,P2) BYTE;
73 2 1 END;
74 1 1 CONVASC: PROCEDURE (P1) EXTERNAL;
75 2 1 DCL P1 BYTE;
76 2 1 END;
77 1 1 DEBLK: PROCEDURE EXTERNAL;

```



```

98 1 = DCL      EM      LIT '3',      /* EMULATOR MODULE NUMBER */
      =      MAX$MSG$OUT LIT '4',      /* MAX. NUMBER OF MSGS. */
      =      MAX$MSG$IN  LIT '4',      /* IN AND OUT MODULES */
      =
      = NUM$PER$TASK  BYTE PUBLIC,      /* NUMBER OF PERIODIC TASKS */
      = NUM$DEM$TASK  BYTE PUBLIC,      /* NUMBER OF DEMAND TASKS */
      =
      = DATA$BLOCK$PER (8) STRUCTURE /* 8 PERIODIC MODULES MAX */
      = (PERIOD (4)  BYTE, /* PERIOD (CLOCK FORMAT) */
      = NUMBER      BYTE, /* TASK IDENTIFICATION */
      = DELAY       BYTE, /* COMPUTATIONAL DELAY */
      = EXEC$NUM    ADDRESS, /* NUMBER OF EXECUTIONS */
      = TIME$IN$EXEC (4) BYTE, /* TOTAL EXECUTION TIME */
      = NUM$MSG$OUT  BYTE, /* # OF OUTPUT LINKS */
      = MSG$DATA (16) BYTE, /* RECORD (4,4) */
      = INDEX      BYTE ) PUBLIC, /* PERIODIC INDEX */
      =
      = DATA$BLOCK$DEM (8) STRUCTURE /* 8 DEMAND MODULES MAX */
      = (NUMBER      BYTE, /* TASK IDENTIFICATION */
      = DELAY       BYTE, /* COMPUTATIONAL DELAY */
      = EXEC$NUM    ADDRESS, /* NUMBER OF EXECUTIONS */
      = TIME$IN$EXEC (4) BYTE, /* TOTAL EXECUTION TIME */
      = NUM$OF$STAT  BYTE, /* NUMBER OF STATES */
      = STAT$NUM    BYTE, /* CURRENT STATE */
      = NUM$MSG$OUT  BYTE, /* # OF OUTPUT LINKS */
      = MSG$DATA (16) BYTE, /* RECORDS (4,4) */
      = NUM$MSG$IN  BYTE, /* # OF INPUT LINKS */
      = MSG$IN$DATA (24) BYTE ) PUBLIC, /* RECORD (4,6) */
      =
      = TEMP$CLOCK1 (4) BYTE PUBLIC,
      = TEMP$CLOCK2 (4) BYTE PUBLIC,
      = TEMP$CLOCK3 (4) BYTE PUBLIC,
      =
      = MSG$HEADER (4) BYTE PUBLIC,
      = MSG$TEXT   (5) BYTE PUBLIC,
      =

```

248


```

$EJECT
$ INCLUDE (:F1:EMUMSG.TRE)

/*****
*****
*****: EMULA2 LOCAL VARIABLES. USED FOR CONTROLLING THE EMULATION.
*****
*****/

99 1 DCL STATE BYTE EXTERNAL, /* STATE OF INPUT PROCESS */
    WRITING BYTE EXTERNAL, /* ENTRY STATE FLAG */
    COUNTER BYTE EXTERNAL, /* INDEX FOR COUNTER PROCEDURE */
    EMULATION$TIME ADDRESS EXTERNAL, /* TIME LIMIT FOR EMULATION */
    EMU$CLOCK1 (4) BYTE EXTERNAL, /* CLOCK SAVER NUMBER 1 */
    EMU$CLOCK2 (4) BYTE EXTERNAL, /* CLOCK SAVER NUMBER 2 */
    EMU$CLOCK3 (4) BYTE EXTERNAL, /* CLOCK SAVER NUMBER 3 */
    DATA$SET$MSG (4) BYTE EXTERNAL, /* HEADER OF NETWORK MSG DATA */
    OUT$TIME (14) BYTE EXTERNAL, /* OUTPUT TIME MESSAGE */
    READY BYTE EXTERNAL, /* READY FOR EMULATION (FLAG) */
    NUM$CPTR$USED BYTE EXTERNAL, /* NUMBER OF COMPUTERS REQUESTED */
    CPTR$IN$USE BYTE EXTERNAL, /* CURRENT COMPUTER IN USE */
    TASK$NUMBER BYTE EXTERNAL, /* NUMBER OF TASK INFO RECEIVED */
    MSG$NUM BYTE EXTERNAL, /* # OF MSG DATA RECEIVED */
    PER$TASK$NUM BYTE EXTERNAL, /* # OF PERIODIC TASK DATA REC. */
    DEM$TASK$NUM BYTE EXTERNAL, /* # OF DEMAND TASK DATA REC. */
    NPT BYTE EXTERNAL, /* # PERIODIC TASK TO SIMULATE */
    NMO BYTE EXTERNAL, /* NUMBER OF MSG TO BE SENDED */
    MO BYTE EXTERNAL, /* NUMBER OF MSG PROCESSED */
    NDT BYTE EXTERNAL, /* # OF DEMAND TASK TO SIMULATE */
    NMI BYTE EXTERNAL, /* NUMBER OF MSG TO BE RECEIVED */
    MI BYTE EXTERNAL, /* NUMBER OF MSG RECEIVED ALREADY */

100 1 DECLARE /* THE FOLLOWING MESSAGES TEXTS */
    MSG00(*) BYTE DATA (1, 'EMULATOR...'),
    MSG01(*) BYTE DATA (0, 'NUMBER OF COMPUTERS NEEDED ?'),
    MSG02(*) BYTE DATA (1, 'CONSOLE REFUSED !!'),

```

```

MSG03(*) BYTE DATA (1, 'CONSOL RELEASED !!'), PERIOD (IN MS) ?.'),
MSG16(*) BYTE DATA (0, 'PERIODIC DATA.', 0AH, 0DH, DELAY (IN MS) ?.'),
MSG18(*) BYTE DATA (0, 'DEMAND DATA.', 0AH, 0DH, PERIOD (IN MS) ?.'),
MSG10(*) BYTE DATA (1, '?????'),
MSG22(*) BYTE DATA (1, ' '),
MSG33(*) BYTE DATA (1, 'READY TO EMULATE ( G? ) !!'),
MSG39(*) BYTE DATA (1, 'INVALID COMMAND!!'),
MSG40(*) BYTE DATA (1, 'END EMULATION .. STATISTICS AVAILABLE'),
MSG41(*) BYTE DATA (4, 3, 1, 5),
MSG42(*) BYTE DATA (4, 3, 2, 5),
MSG43(*) BYTE DATA (4, 3, 3, 5),
MSG44(*) BYTE DATA (4, 3, 4, 5),
MSG45(*) BYTE DATA (4, 3, 5, 4),
MSG46(*) BYTE DATA (4, 3, 6, 4),
MSG47(*) BYTE DATA (4, 3, 7, 4),
MSG48(*) BYTE DATA (12, 3, 1, 5),
MSG49(*) BYTE DATA (12, 3, 2, 5),
MSG50(*) BYTE DATA (12, 3, 3, 5),
MSG51(*) BYTE DATA (12, 3, 4, 5),
MSG52(*) BYTE DATA (12, 3, 5, 4),
MSG53(*) BYTE DATA (12, 3, 6, 4),
MSG54(*) BYTE DATA (12, 3, 7, 4),
MSG55(*) BYTE DATA (20, 3, 1, 5),
MSG56(*) BYTE DATA (20, 3, 2, 5),
MSG57(*) BYTE DATA (20, 3, 3, 5),
MSG58(*) BYTE DATA (20, 3, 4, 5),
MSG59(*) BYTE DATA (20, 3, 5, 4),
MSG60(*) BYTE DATA (20, 3, 6, 4),
MSG61(*) BYTE DATA (20, 3, 7, 4),
MSG62(*) BYTE DATA (2, 3, 14, 4),
MSG63(*) BYTE DATA (10, 3, 14, 4),
MSG64(*) BYTE DATA (18, 3, 14, 4),
MSG65(*) BYTE DATA (0, 'EMULATION LASTED : '),
MSG66(*) BYTE DATA (1, 'EMULATION INITIATED '),
MSG67(*) BYTE DATA (1, 'UNABLE TO RELATE OUTPUT MESSAGES'),
MSG68(*) BYTE DATA (1, 'NOT READY FOR EMULATION.'),

```

```

MSG69(*) BYTE DATA(1, 'INVALID TASK IDENTIFICATION'),
MSG70(*) BYTE DATA(1, 'INVALID COMPUTER NUMBER'),
MSG71(*) BYTE DATA(1, 'NO LINK MESSAGES III'),
MSG80(*) BYTE DATA(0, 'ENTER NEW DATA ? ( Y/N' ) .');
=
$ INCLUDE (:F1:EMULA2.EXT)
=
RECEIVE$INPUT: PROCEDURE EXTERNAL;
101 1
102 2
END;
=
WRITE$ACK$RECEIVED: PROCEDURE EXTERNAL;
103 1
104 2
END;
=
SEND$INFO: PROCEDURE (P1,P2,P3,P4,P5) BYTE EXTERNAL;
105 1
106 2
107 2
DCL P3 ADDRESS, (P1,P2,P4,P5) BYTE;
END;

```



```

122      $EJECT
123      MSGENT4: PROCEDURE EXTERNAL;
124      END;
125      SET$MODS: PROCEDURE EXTERNAL;
126      END;
127      EXIT: PROCEDURE EXTERNAL;
128      END;

/*
*****
*
GET$TASK: GET COMPUTER INDEX GIVEN TASK ID
*****
*/
128      GET$TASK: PROCEDURE (ID) BYTE;
129      DCL ID BYTE;

130      DO B2 = 1 TO TASK$NUMBER ;
131      IF ID$TBL(B2-1).NUMBER = ID THEN RETURN B2-1;
132      END;
133      RETURN 0FFH;
134      FND GET$TASK;
135

```



```

136      $EJECT
137      /*
138      *****
139      PRINT$TIME: PRINT CONSUMED EMULATION TIME ON LINE PRINTER
140      *****
141      */
142      PRINT$TIME: PROCEDURE;
143
144      IF MSGDATA(0) THEN DO; /* CONNECTED TO LINE PRINTER */
145          CALL PAGE(EM);
146          CALL WRITE(.MSG65,LENGTH(MSG65),EM);
147          CALL PRINT$SYNC(.MSG65,LENGTH(MSG65),EM);
148          OUT$TIME(0) = 1;
149          CALL NUMOUT(DOUBLE(EMU$CLOCK3(0)),16,.,.,OUT$TIME(1),3);
150          CALL NUMOUT(DOUBLE(EMU$CLOCK3(1)),16,.,.,OUT$TIME(4),3);
151          CALL NUMOUT(DOUBLE(EMU$CLOCK3(2)),16,.,.,OUT$TIME(7),3);
152          CALL NUMOUT(DOUBLE(EMU$CLOCK3(3)),16,.,.,OUT$TIME(10),3);
153          OUT$TIME(13) = 'H';
154          CALL WRITE(.OUT$TIME,LENGTH(OUT$TIME),EM);
155          CALL PRINT$SYNC(.OUT$TIME,LENGTH(OUT$TIME),EM);
156          CALL PAR$RELEASE;
157          DO B1 = 1 TO NUM$CPTR$USED;
158              CALL UPDSTAT(SHL(B1-1,3)+1,1);
159              END;
160          RETURN;
161      END;
162      ELSE CALL PAR$LINK$REQ(EM);
163      RETURN;
164      END PRINT$TIME;

```



```

159 1 160 2 161 3 162 4 163 5 164 6 165 7 166 8 167 9 168 10 169 11 170 12 171 13 172 14 173 15 174 16 175 17 176 18 177 19 178 20 179 21 180 22 181 23 182 24 183 25 184 26 185 27 186 28 187 29 188 30 189 31
$EJECT
/*
*****
SET$OUT$MSG: SETS INFORMATION ABOUT INTERNAL MESSAGE RECEIVERS
*****
*/
SET$OUT$MSG: PROCEDURE PUBLIC;
DCL RCPT BYTE;

READY = FALSE;
IF MSG$NUM = 0 THEN DO;
  READY = TRUE;
  CALL PRINT$LINKED(.MSG71,LENGTH(MSG71),EM,29);
  RETURN;
END;

DO B1 = 0 TO MSG$NUM - 1 ;
  B3 = GET$TASK(MSG$TBL(B1).SINK);
  IF B3 = 0FFH THEN DO;
    CALL PRINT$SYNC(.MSG67,LENGTH(MSG67),EM);
    CALL PRINT$LINKED(.MSG68,LENGTH(MSG68),EM,29);
    RETURN;
  END;

  RCPT = ID$TBL(B3).COMPT;
  B3 = GET$TASK(MSG$TBL(B1).SOURCE);
  DATA$SET$MSG(RMN) = ID$TBL(B3).COMPT + 2 ;
  IF ID$TBL(B3).KIND = 1 /* PERIODIC */
    THEN B2 = SEND$INFO(6,10,.SET$OUT$MSG,6,11);
    ELSE B2 = SEND$INFO(11,10,.SET$OUT$MSG,6,19);
  MSGDATA(0) = ID$TBL(B3).INDEX;
  MSGDATA(1) = MSG$TBL(B1).INDEX;
  MSGDATA(2) = RCPT + 1 ;
  MSGDATA(3) = MSG$TBL(B1).SINK;
  MSGDATA(4) = MSG$TBL(B1).NUMBER;
  MSGDATA(5) = MSG$TBL(B1).LENT;
  END;

  READY = TRUE;

```

190 2 CALL PRINT\$LINKED(.MSG33,LENGTH(MSG33),EM,29);
191 2 END SET\$OUT\$MSG;

```

192  $EJECT
193  /*
195  *
196  *
197  *
198  *
199  *
200  *
201  *
202  *
203  *
204  *
205  *
206  *
207  *
208  *
209  *
210  *
211  *
212  *
213  *
214  *
215  *
216  *
217  *
218  *
219  *
220  *
221  *
222  *
223  *
224  *

```

SUBS\$DATA: SUBSTITUTE INPUT DATA FOR EMULATION
 SUBS\$DATA: PROCEDURE PUBLIC;

```

193  IF NOT GET$CHAR THEN DO;
195      CALL PRINT$SYNC(.MSG00,LENGTH(MSG00),EM);
196      CALL PRINT$SYNC(.MSG01,LENGTH(MSG01),EM);
197      CALL CON$INPUT$REQ(EM,TRUE);
198      STATE = 0;
199      RETURN;
200      END;
201  IF BL = 'T' THEN DO;
202      IF NOT GET$NUM THEN DO;
203          CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
204          RETURN;
205          END;
206      B3 = GET$TASK(BL);
207      IF B3 = 0FFH THEN DO;
208          CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
209          RETURN;
210          END;
211      TASK$NUMBER = B3 + 1;
212      DATA$SET$MSG(RMN) = ID$TBL(B3).COMPT + 2;
213      CPTR$IN$USE = ID$TBL(B3).CPT$CNT;
214      MSG$NUM = ID$TBL(B3).MSG$CNT;
215      IF ID$TBL(B3).KIND = 1 THEN DO;
216          PER$TASK$NUM = ID$TBL(B3).INDEX + 1;
217          CALL PRINT$SYNC(.MSG16,LENGTH(MSG16),EM);
218          STATE = 7;
219          END;
220      ELSE DO;

```

```
225      DEM$TASK$NUM = ID$TBL(B3).INDEX + 1 ;
226      CALL PRINT$SYNC(.MSG18,LENGTH(MSG18),EM);
227      STATE = 16;
228      END;
229      CALL CON$INPUT$REQ(EM,TRUE);
230      RETURN;
231      END;
232      IF BL = 'P' THEN DO;
233      CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,26);
234      RETURN;
235      END;
236      B1 = B1 - 1 ;
237      IF NOT GETNUM THEN DO;
238      CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
239      RETURN;
240      END;
241      CPTR$IN$USE = BL - 1 ;
242      CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,01);
243      RETURN;
244      END SUBS$DATA;
245
246
```

```

247 1  $EJECT
248 2  /*
249 2  *****
250 3  *****
251 3  *****
252 3  *****
253 3  *****
254 2  *****
255 2  *****
256 2  *****
257 2  *****
258 3  *****
259 3  *****
260 3  *****
261 3  *****
262 2  *****
263 2  *****
264 3  *****
265 3  *****
266 3  *****
267 2  *****
268 2  *****
269 2  *****
270 3  *****
271 3  *****
272 3  *****
273 3  *****
274 2  *****
275 2  *****
276 2  *****
277 3  *****

MOVE$TASK: MOVE TASK LOCATION FROM ONE COMPUTER TO ANOTHER
*****
*/
MOVE$TASK : PROCEDURE PUBLIC;
  DCL (TSK, REC, KND, NUM, OLD, NEW) BYTE;

  IF NOT GETNUM THEN DO;
    CALL PRINT$LINKED(.MSG39, LENGTH(MSG39), EM, 29);
    RETURN;
  END;

  TSK = BL;
  B3 = GET$TASK(TSK);
  IF B3 = 0FFH THEN DO;
    CALL PRINT$SYNC(.MSG69, LENGTH(MSG69), EM);
    CALL PRINT$LINKED(.MSG39, LENGTH(MSG39), EM, 29);
    RETURN;
  END;

  IF NOT GETNUM THEN DO;
    CALL PRINT$LINKED(.MSG39, LENGTH(MSG39), EM, 29);
    RETURN;
  END;

  REC = BL;
  IF REC = ID$TBL(B3).CPT$CNT OR
  REC < 1 OR REC > 3 THEN DO;
    CALL PRINT$SYNC(.MSG70, LENGTH(MSG70), EM);
    CALL PRINT$LINKED(.MSG39, LENGTH(MSG39), EM, 29);
    RETURN;
  END;

  KND = ID$TBL(B3).KIND;
  REC = SHL(REC-1, 3) + 4;
  DO CASE ID$TBL(B3).CPT$CNT - 1;
    A1 = .MSG45;

```

```

278      A1 = .MSG52;
279      A1 = .MSG59;
280      END;
281      B2 = SEND(.A1);
282      MSGDATA(0) = KND;
283      MSGDATA(1) = TSK;
284      MSGDATA(2) = REC;

285      OLD = ID$TBL(B3).COMPT + 1;
286      NEW = REC - 3;
287      DO B1 = 1 TO MSG$NUM;
288          IF MSG$TBL(B1-1).SINK = TSK THEN DO;
289              NUM = MSG$TBL(B1-1).SOURCE;
290              B3 = GET$TASK(NUM);
291              IF B3 <> 0FFH THEN DO;
292                  DO CASE ID$TBL(B3).CPT$CNT - 1;
293                      A1 = .MSG47;
294                      A1 = .MSG54;
295                      A1 = .MSG61;
296                      END;
297                      B2 = SEND(A1);
298                      MSGDATA(0) = NUM;
299                      MSGDATA(1) = OLD;
300                      MSGDATA(2) = NEW;
301                      END;
302                      END;
303                      END;
304                      END;
305                      END;
306                      END;
307                      END;

```



```

308      $EJECT
309      /*
310      *****
311      EMU$COUNT: EMULATION COUNTER
312      *****
313      */
314      EMU$COUNT: PROCEDURE;
315
316      DISABLE;
317      CALL PRIORITY(COUNTER);
318      ENABLE;
319      RETURN;
320      END EMU$COUNT;
321
322      /*
323      *****
324      EMU$COUNTER: COUNTS UP UNTIL END OF EMULATION
325      *****
326      */
327      EMU$COUNTER: PROCEDURE;
328      DCL STOP (4) BYTE DATA(EM, EM, 30, 4);
329
330      IF EMULATION$TIME = 0 THEN DO;
331          B2 = SEND(.STOP);
332          CALL REMPRIOR(COUNTER);
333          RETURN;
334      END;
335
336      IF EMULATION$TIME < 50 THEN EMULATION$TIME = 50;
337      EMULATION$TIME = EMULATION$TIME - 50;
338      IF NOT SET$CDC(0D203H, .EMU$COUNT) /* SET FOR 50 MS */
339          THEN CALL ERROR(.EMU$COUNTER, 6, 30);
340      RETURN;
341      END EMU$COUNTER;

```

```

329 $EJECT
330 /*
331 *****
332 *****
333 *****
334 *****
335 *****
336 *****
337 *****
338 *****
339 *****
340 *****
341 *****
342 *****
343 *****
344 *****
345 *****
346 *****
347 *****
348 *****
349 *****
351 *****
352 *****
353 *****

START$EMU: INITIALIZE EMULATION
*****
*/
START$EMU: PROCEDURE PUBLIC;

    IF NOT READY THEN DO;
        CALL PRINT$LINKED(.MSG68,LENGTH(MSG68),EM,29);
        RETURN;
    END;
    DO B1 = 1 TO NUM$CPTR$USED ;
        DO CASE B1-1;
            B2 = SEND(.MSG62);
            B2 = SEND(.MSG63);
            B2 = SEND(.MSG64);
        END;
    END;
    CALL PRINT$SYNC(.MSG66,LENGTH(MSG66),EM);
    IF EMULATION$TIME = 0 THEN DO;
        CALL COPY$CLOCK(.EMU$CLOCK1);
        RETURN;
    END;
    COUNTER = ENTER$PRIOR(.EMU$COUNTER);
    IF NOT SET$CDC(0D203H,.EMU$COUNT) /* SET 50 MS */
        THEN CALL ERROR(.START$EMU,6,31);
    CALL COPY$CLOCK(.EMU$CLOCK1);
    RETURN;
END START$EMU;

```

```

354 $EJECT
355 /*
356 *****
357 SEND$WRITE$MSG: SENDS REQUEST TO OUTPUT INPUT DATA
358 *****
359 */
360 SEND$WRITE$MSG: PROCEDURE (H,PRN,CPT);
361   DCL (H,PRN,CPT) BYTE;
362
363   IF NUM$CPTR$USED = 0 THEN DO;
364     CALL PRINT$LINKED(.MSG10,LENGTH(MSG10),EM,29);
365     RETURN;
366   END;
367
368   IF CPT > NUM$CPTR$USED THEN DO;
369     CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
370     RETURN;
371   END;
372
373   A1 = .MSG41 + DOUBLE(SHL(PRN,2));
374   DO CASE CPT;
375     DO B2 = 1 TO NUM$CPTR$USED;
376       B3 = SEND(A1);
377       MSGDATA(0) = H ;
378       A1 = A1 + 28;
379       END;
380     DO; B3 = SEND(A1);
381       MSGDATA(0) = H ;
382       END;
383     DO; A1 = A1 + 28;
384       B3 = SEND(A1);
385       MSGDATA(0) = H ;
386       END;

```

387	3	END;
388	2	WRITING = TRUE;
389	2	CALL CON\$RELEASE;
390	2	RETURN;
391	2	END SEND\$WRITE\$MSG\$;

```

392 $EJECT
393 /*
*****
WRITE$DATA: WRITES INPUTED DATA ON LINE PRINTER
*****
*/
WRITE$DATA: PROCEDURE PUBLIC;
DCL (HARD$COPY,PRN,CPT) BYTE;

HARD$COPY = FALSE;
PRN = 2 ; CPT = 0 ;
DO WHILE 1; /* FOREVER */
IF NOT GET$CHAR THEN DO;
CALL SEND$WRITE$MSG$(HARD$COPY,PRN,CPT);
RETURN;
END;
IF BL = 'P' THEN PRN = 0;
IF BL = 'D' THEN PRN = 1;
IF BL = 'S' THEN PRN = 3;
IF BL > '0' AND BL < '4' THEN CPT = BL - '0' ;
IF BL = 'H' THEN HARD$COPY = TRUE;
END;
END WRITE$DATA;
/*
*****
DISCONNECT$CRT: CRT CONNECTION REQUEST HAS BEEN REFUSED
*****
*/
DISCONNECT$CRT: PROCEDURE;
CALL PRINT$ASYNC(.MSG03,LENGTH(MSG03));
RETURN;
END DISCONNECT$CRT;

```



```

$EJECT
/*
*****
STOP$EMU: STOP A EMULATION
*****
*/
443 1  STOP$EMU: PROCEDURE;
444 2  DCL STOP$PER1 (4) BYTE DATA(2,3,17,4),
      STOP$PER2 (4) BYTE DATA(10,3,17,4),
      STOP$PER3 (4) BYTE DATA(18,3,17,4);

445 2  CALL COPY$CLOCK(.EMU$CLOCK2);
446 2  CALL VECTOR$SUB(.EMU$CLOCK2,.EMU$CLOCK1,.EMU$CLOCK3);
447 2  CALL PAR$LINK$REQ(EM);

448 2  DO B1 = 1 TO NUM$CPTR$USED;
449 3    DO CASE B1-1;
450 4      B2 = SEND(.STOP$PER1);
451 4      B2 = SEND(.STOP$PER2);
452 4      B2 = SEND(.STOP$PER3);
453 4    END;
454 3    CALL UPDSTAT(SHL(B1-1,3)+1,1);
455 3  END;
456 2  CALL PRINT$LINKED(.MSG40,LENGTH(MSG40),3,29);
457 2  RETURN;
458 2  END STOP$EMU;

```

```

$EJECT
/*
*****
MSGENT3: MESSAGE ENTRY POINT FOR DATA ENTRY MODULE
*****
*/
MSGENT3: PROCEDURE;

    IF MSG(MN) = 0 THEN DO;
        CALL INIT$EMU;
        RETURN;
    END;

    IF MSG(MN) > 31 OR MSG(MN) < 20 THEN DO;
        CALL ILLEGALMSG;
        RETURN;
    END;

    DO CASE MSG(MN) - 20;
        CALL RECEIVE$INPUT; /* MN 20 */
        CALL CHECK$SE$CONNECTION; /* MN 21 */
        CALL DISCONNECT$CRT; /* MN 22 */
        CALL ILLEGALMSG;
        CALL ILLEGALMSG;
        CALL ILLEGALMSG;
        CALL WRITE$ACK$RECEIVED; /* MN 26 */
        CALL ILLEGALMSG; /* MN 28 */
        CALL PRINT$TIME; /* MN 30 */
        CALL ILLEGALMSG; /* MN 31 */
        CALL STOP$EMU;
        CALL CON$LINK$REQ(EM);
    END;

    RETURN;
END MSGENT3;

```

SUBJECT

✱

MAIN PROGRAM : EXECUTED UNDER ISIS. STABLISH MODULES INTO THE SYSTEM.

* /

```

486 1 CALL CLEAR$MOD;
487 1 CALL ENTER$MOD(3,.MSGENT3);
488 1 CALL ENTER$MOD(4,.MSGENT4);
489 1 CALL ENTER$MOD(12,.MSGENT4);
490 1 CALL ENTER$MOD(20,.MSGENT4);
491 1 CALL EXIT;

```

492 1 END FMULA2;

MODULE INFORMATION:

```

CODE AREA SIZE      3275D
VARIABLE AREA SIZE  1226D
MAXIMUM STACK SIZE   10D
      857 LINES READ
      0 PROGRAM ERROR(S)
      = 00CBH
      = 04CAH
      = 000AH

```

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EMULA3
 NO OBJECT MODULE REQUESTED
 COMPILER INVOKED BY: PIMB0 :F1:EMULA3.SRC NOOBJECT PAGLENGTH(39) PAGEWIDTH(90)

```

1      EMULA3:
/*****
*****
***** : THIS PROGRAM WRITES DATA AND STATISTICS
***** ON THE SYSTEM CONSOLE OR LINE PRINTER.
*****
**/
DO;

$NOLIST
$ INCLUDE (:F1:EMUMSG.TWO)
/*****
*****
***** : EMULA3 LOCAL DATA. CONTAINS THE FORMATS FOR THE DISPLAYS.
*****
**/

99      1      DECLARE /* THE FOLLOWING DATA */
          MSG00(*) BYTE DATA(0,'EMULATION ELAPSED :'),
          MSG01(*) BYTE DATA(1,''),
          MSG02(*) BYTE DATA(0,'DATA COMPUTER :'),
          MSG03(*) BYTE DATA(0,'PERIODIC TASK :'),
          MSG04(*) BYTE DATA(0,'TASK #'),
          MSG05(*) BYTE DATA(0,'-----'),
          MSG06(*) BYTE DATA(0,'PERIOD'),
          MSG07(*) BYTE DATA(0,'-----'),
          MSG08(*) BYTE DATA(0,'DELAY'),
          MSG09(*) BYTE DATA(0,'-----'),
          MSG10(*) BYTE DATA(1,'MSG OUT'),
          MSG11(*) BYTE DATA(0,'-----');

```

```

MSG12(*) BYTE DATA(0), # STATES',
MSG13(*) BYTE DATA(0), -----',
MSG14(*) BYTE DATA(0), MSG IN',
MSG15(*) BYTE DATA(0), -----',
MSG16(*) BYTE DATA(0), # EXECS',
MSG17(*) BYTE DATA(0), -----',
MSG18(*) BYTE DATA(0), T. IN EXEC',
MSG19(*) BYTE DATA(0), -----',
MSG20(*) BYTE DATA(1), AVG.TIME',
MSG21(*) BYTE DATA(1), -----',
MSG22(*) BYTE DATA(0), NUMBER',
MSG23(*) BYTE DATA(0), -----',
MSG24(*) BYTE DATA(0), # RECEV.',
MSG25(*) BYTE DATA(0), -----',
MSG26(*) BYTE DATA(0), TOTAL DELAY',
MSG27(*) BYTE DATA(0), -----',
MSG28(*) BYTE DATA(0), DEMAND TASK:',
MSG29(*) BYTE DATA(1), MESSAGES:',

MSG30(10) BYTE, /* TASK */
MSG31(11) BYTE, /* PERIOD */
MSG32(10) BYTE, /* DELAY */
MSG33(15) BYTE, /* MSGOUT */
MSG34(12) BYTE, /* # STAT */
MSG35(15) BYTE, /* MSGIN */
MSG36(11) BYTE, /* # EXEC */
MSG37(14) BYTE, /* TIME */
MSG38(12) BYTE, /* AVG. T */
MSG39(13) BYTE, /* NUMBER */
MSG40(12) BYTE, /* RECEIV */
MSG41(15) BYTE, /* T.DELAY */
MSG42    BYTE,

PRN$SEQUENCE BYTE, /* INDEX FOR DATA PRINT SEQUENCE */
HARD$COPY    BYTE, /* HARD COPY FLAG */
MOD$ID       BYTE, /* MODULE IDENTIFICATION */

```

272


```

$EJECT
/*
*****
WRT: WRITES DATA ON CRT OR LINE PRINTER.
*****
*/
100 1 WRT: PROCEDURE (ADR, LEN, SEN);
101 2   DCL ADR ADDRESS, (LEN, SEN) BYTE;

102 2   IF HARD$COPY THEN CALL WRITE(ADR, LEN, SEN);
104 2       ELSE CALL PRINT$SYNC(ADR, LEN, SEN);
105 2   RETURN;
106 2   END WRT;
/*
*****
WRT$LINKED: WRITES DATA ON CRT OR LINE PRINTER AND REQUEST
              FOR TELL BACK CODE.
*****
*/
107 1 WRT$LINKED: PROCEDURE (ADR, LEN, SEN, TBC);
108 2   DCL ADR ADDRESS, (LEN, SEN, TBC) BYTE;

109 2   IF HARD$COPY THEN CALL WRITE$LINKED(ADR, LEN, SEN, TBC);
111 2       ELSE CALL PRINT$LINKED(ADR, LEN, SEN, TBC);
112 2   RETURN;
113 2   END WRT$LINKED;

```

274

```

$EJECT
/*
*****
PRN$DATA$PER: PRINT PERIODIC TASK DATA
*****
*/
120 PRN$DATA$PER: PROCEDURE (INDEX);
121   DCL INDEX BYTE;
122   DCL ADR$ELE ADDRESS,
      (ELE BASED ADR$ELE) BYTE;

123   CALL SET$BUFFERS;

124   ADR$ELE = .DATA$BLOCK$PER(INDEX).NUMBER;
125   CALL NUMOUT(DOUBLE(ELE),10,'',.MSG30(5),3);
126   CALL WRT(.MSG30,LENGTH(MSG30),MOD$ID);

127   ADR$ELE = ADR$ELE - 2 ; BU = ELE ;
129   ADR$ELE = ADR$ELE + 1 ; BL = ELE ;
131   CALL NUMOUT(A4,10,'',.MSG31(5),4);
132   CALL WRT(.MSG31,LENGTH(MSG31),MOD$ID);

133   ADR$ELE = ADR$ELE + 2;
134   CALL NUMOUT(DOUBLE(ELE),10,'',.MSG32(5),3);
135   CALL WRT(.MSG32,LENGTH(MSG32),MOD$ID);

136   ADR$ELE = .DATA$BLOCK$PER(INDEX).NUM$MSG$OUT;
137   B2 = ELE ; B3 = 4 ;
139   ADR$ELE = ADR$ELE + 3; /* POINT TO FIRST MSG NUMBER */
140   DO WHILE B2 > 0;
141     CALL NUMOUT(DOUBLE(ELE),10,'',.MSG33(B3),2);
142     ADR$ELE = ADR$ELE + 4; /* POINT TO NEXT MSG NUMBER */
143     B3 = B3 + 3;
144     B2 = B2 - 1;
145   END;
146   CALL WRT(.MSG33,LENGTH(MSG33),MOD$ID);

```

```

$EJECT
/*
*****
PRN$DATA$DEM: PRINT DEMAND DATA
*****
*/
149 1 PRN$DATA$DEM: PROCEDURE (INDEX);
150 2   DCL INDEX BYTE;
151 2   DCL ADR$ELE ADDRESS,
      (ELE BASED ADR$ELE) BYTE;

152 2   CALL SET$BUFFERS;

153 2   ADR$ELE = .DATA$BLOCK$DEM(INDEX).NUMBER;
154 2   CALL NUMOUT(DOUBLE(ELE),10,'',.MSG30(5),3);
155 2   CALL WRT(.MSG30,LENGTH(MSG30),MOD$ID);

156 2   ADR$ELE = ADR$ELE + 1;
157 2   CALL NUMOUT(DOUBLE(ELE),10,'',.MSG32(5),3);
158 2   CALL WRT(.MSG32,LENGTH(MSG32),MOD$ID);

159 2   ADR$ELE = ADR$ELE + 7;
160 2   CALL NUMOUT(DOUBLE(ELE),10,'',.MSG34(7),3);
161 2   CALL WRT(.MSG34,LENGTH(MSG34),MOD$ID);

162 2   ADR$ELE = .DATA$BLOCK$DEM(INDEX).NUM$MSG$IN;
163 2   B2 = ELE ;
164 2   B3 = 4;
165 2   ADR$ELE = ADR$ELE + 1;
166 2   DO WHILE B2 > 0;
167 3     CALL NUMOUT(DOUBLE(ELE),10,'',.MSG35(B3),2);
168 3     B2 = B2 - 1;
169 3     B3 = B3 + 3;
170 3     ADR$ELE = ADR$ELE + 6;
171 3     END;

```

```
172 2      CALL WRT(.MSG35,LENGTH(MSG35),MOD$ID);
173 2      ADR$ELE = .DATA$BLOCK$DEM(INDEX).NUM$MSG$OUT;
174 2      B2 = ELE;
175 2      B3 = 4;
176 2      ADR$ELE = ADR$ELE + 3; /* POINT TO FIRST MSF NUMBER */
177 2      DO WHILE B2 > 0;
178 3          CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG33(B3),2);
179 3          ADR$ELE = ADR$ELE + 4; /* POINT TO NEXT MSG NUMBER */
180 3          B2 = B2 - 1;
181 3          B3 = B3 + 3;
182 3      END;
183 2      CALL WRT(.MSG33,LENGTH(MSG33),MOD$ID);
184 2      RETURN;
185 2      END PRN$DATA$DEM;
```

```

186      $EJECT
187      /*
188      *****;
189      PRN$ORDER: PRINTS SEQUENTIAL NUMBER ON LINE PRINTER
190      *****
191      */
192      PRN$ORDER: PROCEDURE (P1);
193      DCL P1 BYTE;
194      DO CASE P1;
195          CALL WRT( .MSG59, LENGTH(MSG59), MOD$ID);
196          CALL WRT( .MSG51, LENGTH(MSG51), MOD$ID);
197          CALL WRT( .MSG52, LENGTH(MSG52), MOD$ID);
198          CALL WRT( .MSG53, LENGTH(MSG53), MOD$ID);
199          CALL WRT( .MSG54, LENGTH(MSG54), MOD$ID);
200          CALL WRT( .MSG55, LENGTH(MSG55), MOD$ID);
          CALL WRT( .MSG56, LENGTH(MSG56), MOD$ID);
          CALL WRT( .MSG57, LENGTH(MSG57), MOD$ID);
          CALL WRT( .MSG58, LENGTH(MSG58), MOD$ID);
          END;
      RETURN;
      END PRN$ORDER;

```



```

$EJECT
/*
*****
PRN$HEAD$PER: PRINT HEADER OF PERIODIC TASK
*****
*/
PRN$HEAD$PER: PROCEDURE;

201 1
202 2 CALL WRT(.MSG03,LENGTH(MSG03),MOD$ID);
203 2 CALL PRN$ORDER(NUM$PER$TASK);
204 2 CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
205 2 CALL WRT(.MSG04,LENGTH(MSG04),MOD$ID);
206 2 CALL WRT(.MSG06,LENGTH(MSG06),MOD$ID);
207 2 CALL WRT(.MSG08,LENGTH(MSG08),MOD$ID);
208 2 CALL WRT(.MSG10,LENGTH(MSG10),MOD$ID);
209 2 CALL WRT(.MSG05,LENGTH(MSG05),MOD$ID);
210 2 CALL WRT(.MSG07,LENGTH(MSG07),MOD$ID);
211 2 CALL WRT(.MSG09,LENGTH(MSG09),MOD$ID);
212 2 CALL WRT(.MSG11,LENGTH(MSG11),MOD$ID);
213 2 CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
214 2 RETURN;
215 2 END PRN$HEAD$PER;

```

\$EJECT

/*

PRN\$HEAD\$DEM: PRINT HEADER OF DEMAND TASK DATA

*/

PRN\$HEAD\$DEM: PROCEDURE;

```

216 1 PRN$HEAD$DEM: PROCEDURE;
217 2 CALL WRT(.MSG28,LENGTH(MSG28),MOD$ID);
218 2 CALL PRN$ORDER (NUM$DEM$TASK);
219 2 CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
220 2 CALL WRT(.MSG04,LENGTH(MSG04),MOD$ID);
221 2 CALL WRT(.MSG08,LENGTH(MSG08),MOD$ID);
222 2 CALL WRT(.MSG12,LENGTH(MSG12),MOD$ID);
223 2 CALL WRT(.MSG14,LENGTH(MSG14),MOD$ID);
224 2 CALL WRT(.MSG10,LENGTH(MSG10),MOD$ID);
225 2 CALL WRT(.MSG05,LENGTH(MSG05),MOD$ID);
226 2 CALL WRT(.MSG09,LENGTH(MSG09),MOD$ID);
227 2 CALL WRT(.MSG13,LENGTH(MSG13),MOD$ID);
228 2 CALL WRT(.MSG15,LENGTH(MSG15),MOD$ID);
229 2 CALL WRT(.MSG11,LENGTH(MSG11),MOD$ID);
230 2 CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
231 2 RETURN;
232 2 END PRN$HEAD$DEM;

```

```

233 $EJECT
234 /*
235 *****
236 *****
237 *****
238 *****
239 *****
240 *****
241 *****
242 *****
243 *****
244 *****
245 *****
246 *****
247 *****
248 *****
249 *****
250 *****

PRN$PER: WRTS PERIODIC DATA ON LINEPRINTER
*****
*/
PRN$PER: PROCEDURE;
  IF PRN$SEQUENCE = 0 THEN DO;
    CALL PRN$HEAD$PER;
    PRN$SEQUENCE = 1;
    CALL WRT$LINKED(.MSG50, LENGTH(MSG50), MOD$ID, 1);
    RETURN;
  END;
  IF PRN$SEQUENCE > NUM$PER$TASK THEN DO;
    CALL WRT$LINKED(.MSG50, LENGTH(MSG50), MOD$ID, 0);
    RETURN;
  END;
  CALL PRN$DATA$PER(PRN$SEQUENCE-1);
  CALL WRT$LINKED(.MSG50, LENGTH(MSG50), MOD$ID, 1);
  PRN$SEQUENCE = PRN$SEQUENCE + 1;
  RETURN;
END PRN$PER;

```

```

251 $EJECT
252 /*
254 ****
255 ****
256 ****
257 ****
258 ****
259 ****
261 ****
262 ****
263 ****
264 ****
265 ****
266 ****
267 ****
268 ****

PRN$DEM: WRT DEMAND TASK DATA ON LINE PRINTER
****
*/
PRN$DEM: PROCEDURE;
  IF PRN$SEQUENCE = 0 THEN DO;
    CALL PRN$HEAD$DEM;
    PRN$SEQUENCE = 1;
    CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,2);
    RETURN;
  END;
  IF PRN$SEQUENCE > NUM$DEM$TASK THEN DO;
    CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,0);
    RETURN;
  END;
  CALL PRN$DATA$DEM(PRN$SEQUENCE-1);
  CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,2);
  PRN$SEQUENCE = PRN$SEQUENCE + 1;
  RETURN;
END PRN$DEM;

```

\$EJECT

/*

STT\$HEAD: PRINT HEADER OF TASK STATISTICS

*/

STT\$HEAD: PROCEDURE;

269 1

```
270 2 CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
271 2 CALL WRT(.MSG04,LENGTH(MSG04),MOD$ID);
272 2 CALL WRT(.MSG16,LENGTH(MSG16),MOD$ID);
273 2 CALL WRT(.MSG18,LENGTH(MSG18),MOD$ID);
274 2 CALL WRT(.MSG20,LENGTH(MSG20),MOD$ID);
275 2 CALL WRT(.MSG05,LENGTH(MSG05),MOD$ID);
276 2 CALL WRT(.MSG17,LENGTH(MSG17),MOD$ID);
277 2 CALL WRT(.MSG19,LENGTH(MSG19),MOD$ID);
278 2 CALL WRT(.MSG21,LENGTH(MSG21),MOD$ID);
279 2 CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
280 2 RETURN;
281 2 END STT$HEAD;
```

/*

STT\$HEAD\$PER: PRINT HEADER OF PERIODIC TASK STATISTICS

*/

STT\$HEAD\$PER: PROCEDURE;

282 1

```
283 2 CALL WRT(.MSG03,LENGTH(MSG03),MOD$ID);
284 2 CALL PRN$ORDER(NUM$PER$TASK);
285 2 CALL STT$HEAD;
286 2 RETURN;
287 2 END STT$HEAD$PER;
```

```

$EJECT
/*
*****
STT$HEAD$DEM: PRINT HEADER OF DEMAND TASK STATISTICS
*****
*/
STT$HEAD$DEM: PROCEDURE;
  1  CALL WRT(.MSG28,LENGTH(MSG28),MOD$ID);
  2  CALL PRN$ORDER(NUM$DEM$TASK);
  2  CALL STT$HEAD;
  2  RETURN;
  2  END STT$HEAD$DEM;
/*
*****
STT$HEAD$MSG: PFINT GEADER FOR MESSAGE STATISTICS
*****
*/
STT$HEAD$MSG: PROCEDURE;
  1  CALL WRT(.MSG29,LENGTH(MSG29),MOD$ID);
  2  CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
  2  CALL WRT(.MSG22,LENGTH(MSG22),MOD$ID);
  2  CALL WRT(.MSG24,LENGTH(MSG24),MOD$ID);
  2  CALL WRT(.MSG26,LENGTH(MSG26),MOD$ID);
  2  CALL WRT(.MSG20,LENGTH(MSG20),MOD$ID);
  2  CALL WRT(.MSG23,LENGTH(MSG23),MOD$ID);
  2  CALL WRT(.MSG25,LENGTH(MSG25),MOD$ID);
  2  CALL WRT(.MSG27,LENGTH(MSG27),MOD$ID);
  2  CALL WRT(.MSG21,LENGTH(MSG21),MOD$ID);
  2  CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
  2  RETURN;
  2  END STT$HEAD$MSG;
*****

```



```

$EJECT
/*
*****
STT$DATA: PRINT TASK DATA STATISTICS
*****
*/
308 1 STT$DATA: PROCEDURE (ADR$ELE);
309 2   DCL ADR$ELE ADDRESS;
310 2   DCL (ELE BASED ADR$ELE) BYTE,
      ELEA2 ADDRESS,
      (ELEA BASED ADR$ELE) ADDRESS;

311 2   CALL SET$BUFFERS;

312 2   CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG30(5),3);
313 2   CALL WRT(.MSG30,LENGTH(MSG30),MOD$ID);

314 2   ADR$ELE = ADR$ELE + 2;
315 2   CALL NUMOUT(ELEA,10,' ',.MSG36(4),5);
316 2   CALL WRT(.MSG36,LENGTH(MSG36),MOD$ID);
317 2   ELEA2 = ELEA;
318 2   ADR$ELE = ADR$ELE + 4;
319 2   BU = ELE;
320 2   ADR$ELE = ADR$ELE + 1;
321 2   BL = ELE;
322 2   CALL NUMOUT(A4,10,' ',.MSG37(7),5);
323 2   CALL WRT(.MSG37,LENGTH(MSG37),MOD$ID);

324 2   IF ELEA2 = 0 THEN DO;
326 3     CALL WRT(.MSG59,LENGTH(MSG59),MOD$ID);
327 3     RETURN;
328 3     END;
329 2   ELEA2 = A4 / ELFA2;
330 2   CALL NUMOUT(ELEA2,10,' ',.MSG38(5),5);
331 2   CALL WRT(.MSG38,LENGTH(MSG38),MOD$ID);

```

```

334 1      $EJECT
335 2      /*
336 2      ****
337 2      ****
338 2      ****
339 2      ****
340 2      ****
341 2      ****
342 2      ****
343 2      ****
344 2      ****
345 2      ****
346 2      ****
347 2      ****
348 2      ****
349 2      ****
350 2      ****
351 3      ****
352 3      ****
353 3      ****
354 2      ****
355 2      ****
356 2      ****
357 2      ****
358 2      ****

STT$DATA$MSG: PRINT MSG DATA STATISTICS
****
*/
STT$DATA$MSG: PROCEDURE (ADR$ELE);
DCL ADR$ELE ADDRESS;
DCL (ELE BASED ADR$ELE) BYTE,
    ELEA ADDRESS;

CALL NUMOUT(DOUBLE(ELE),10,'',.MSG39(9),3);
CALL WRT(.MSG39,LENGTH(MSG39),MOD$ID);

ADR$ELE = ADR$ELE + 1;
ELEA = DOUBLE(ELE);
CALL NUMOUT(ELEA,10,'',.MSG40(8),3);
CALL WRT(.MSG40,LENGTH(MSG40),MOD$ID);

ADR$ELE = ADR$ELE + 3;
BU = ELE;
ADR$ELE = ADR$ELE + 1;
BL = ELE;
CALL NUMOUT(A4,10,'',.MSG41(7),5);
CALL WRT(.MSG41,LENGTH(MSG41),MOD$ID);

IF ELEA = 0 THEN DO;
    CALL WRT(.MSG59,LENGTH(MSG59),MOD$ID);
    RETURN;
END;

ELEA = A4 / ELEA;
CALL NUMOUT(ELEA,10,'',.MSG38(5),5);
CALL WRT(.MSG38,LENGTH(MSG38),MOD$ID);
RETURN;
END STT$DATA$MSG;

```

```

359      $EJECT
360      /*
361      *****:
362
363      STT$MSG: PRINT STATISTICS OF RECEIVED MESSAGE ON ONE TASK
364      *****
365      */
366      STT$MSG: PROCEDURE (ADR$ELE);
367      DCL ADR$ELE ADDRESS;
368      DCL (ELE BASED ADR$ELE) BYTE,
369           ELE2 BYTE;
370
371      ELE2 = ELE;
372      ADR$ELE = ADR$ELE + 1;
373      DO WHILE ELE2 <> 0;
374          CALL STT$DATA$MSG(ADR$ELE);
375          ADR$ELE = ADR$ELE + 6;
376          ELE2 = ELE2 - 1;
377      END;
378      RETURN;
379      END STT$MSG;

```

AD-A072 578

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/6 9/2

A TACTICAL SYSTEM EMULATOR FOR A DISTRIBUTED MICRO-COMPUTER ARC--ETC(U)

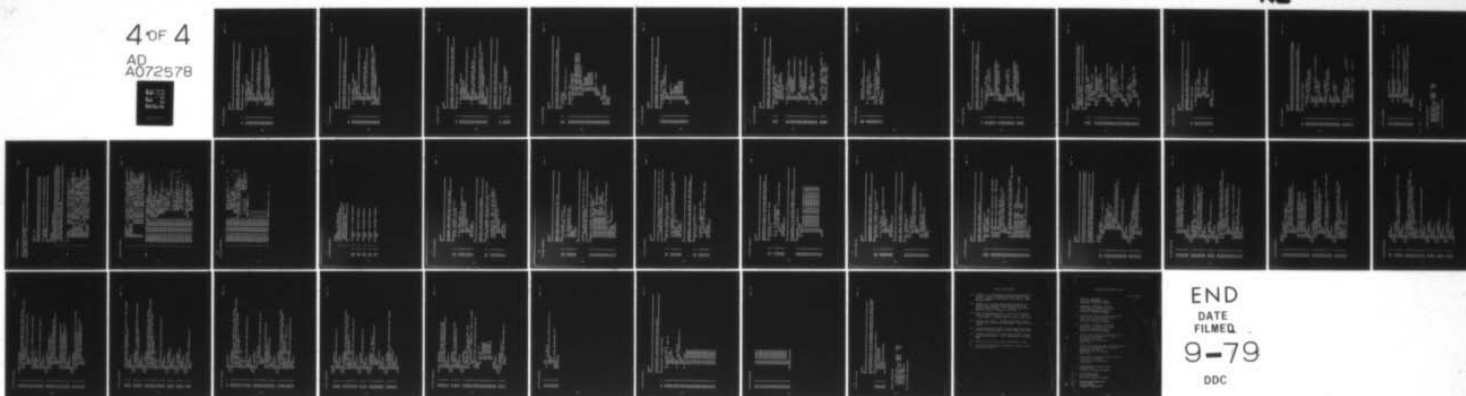
JUN 79 L A GUILLEN

UNCLASSIFIED

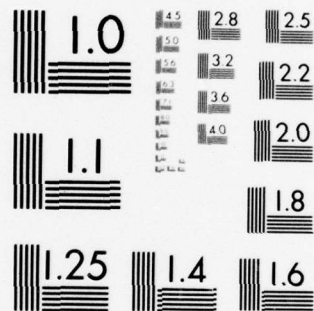
NL

4 OF 4

AD
A072578



END
DATE
FILMED
9-79
DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```

371 1 PRN$STT$PER: PROCEDURE;
372 2 IF PRN$SEQUENCE = 0 THEN DO;
373 3 CALL STT$HEAD$PER;
374 3 PRN$SEQUENCE = 1;
375 3 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,3);
376 3 RETURN;
377 3 END;
378 3 IF PRN$SEQUENCE > NUM$PER$TASK THEN DO;
379 2 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,0);
380 3 RETURN;
381 3 END;
382 3 CALL STT$DATA(.DATA$BLOCK$PER(PRN$SEQUENCE-1).NUMBER);
383 3 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,3);
384 2 PRN$SEQUENCE = PRN$SEQUENCE + 1;
385 2 RETURN;
386 2 END PRN$STT$PER;
387 2
388 2

```



```

389          $EJECT
390          /*
391          ****
392          PRN$STT$DEM: WRITES DEMAND TASK STATISTICS
393          ****
394          */
395          PRN$STT$DEM: PROCEDURE;
396
397          IF PRN$SEQUENCE = 0 THEN DO;
398              CALL STT$HEAD$DEM;
399              PRN$SEQUENCE = 1;
400              CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,4);
401              RETURN;
402          END;
403          IF PRN$SEQUENCE > NUM$DEM$TASK THEN DO;
404              CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,0);
405              RETURN;
406          END;
407          CALL STT$DATA(.DATA$BLOCK$DEM(PRN$SEQUENCE-1).NUMBER);
408          CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,4);
409          PRN$SEQUENCE = PRN$SEQUENCE + 1;
410          RETURN;
411          END PRN$STT$DEM;

```

```

$EJECT
/*
*****
PRN$STT$MSG: WRITES MESSAGES STATISTICS ON LINE PRINTER
*****
*/
PRN$STT$MSG: PROCEDURE;
407 1
408 2 IF PRN$SEQUENCE = 0 THEN DO;
410 3 CALL STT$HEAD$MSG;
411 3 PRN$SEQUENCE = 1;
412 3 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,5);
413 3 RETURN;
414 3 END;
415 2 IF PRN$SEQUENCE > NUM$DEM$TASK THEN DO;
417 3 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,0);
418 3 RETURN;
419 3 END;
420 2 CALL STT$MSG(.DATA$BLOCK$DEM(PRN$SEQUENCE-1).NUM$MSG$IN);
421 2 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,5);
422 2 PRN$SEQUENCE = PRN$SEQUENCE + 1;
423 2 RETURN;
424 2 END PRN$STT$MSG;
/*
*****
RESET$WRITER: RESETS OUTPUT MODULE DEVICE.
*****
*/
RESET$WRITER: PROCEDURE;
425 1
426 2 IF HARD$COPY THEN CALL PAR$RELEASE;
428 2 ELSE CALL CON$RELEASE;
429 2 RETURN;
430 2 END RESET$WRITER;
*****

```

```

*****
$EJECT
/*
*****
PRN: PINTS DATA OR STATISTIC AS INDICATED BY PRN$FLAG
*****
*/
PRN: PROCEDURE;
  DCL P1 BYTE,
  TBMMSG(*) BYTE DATA(3,4,31,4);

  PRN$SEQUENCE = 0;
  DO P1 = 0 TO LAST(PRN$FLAG);
    IF PRN$FLAG(P1) THEN DO;
      CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
      CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
      CALL WRT(.MSG02,LENGTH(MSG02),MOD$ID);
      CALL PRN$ORDER(CPTR$ID+1);
      DO CASE P1;
        CALL PRN$PER;
        CALL PRN$DEM;
        CALL PRN$STT$PER;
        CALL PRN$STT$DEM;
        CALL PRN$STT$MSG;
      END;
      PRN$FLAG(P1) = FALSE;
    RETURN;
  END;

  CALL RESET$WRITER;
  P1 = SEND(.TBMMSG);
  RETURN;
END PRN;
*****

```

```

456 $EJECT
457 /*
459 ****
460 ****
461 ****
462 ****
463 ****
464 ****
465 ****
466 ****
467 ****
468 ****
469 ****
470 ****
471 ****

REC$CO$TBCODE: RECEIVE CONSOLE TELL BACK CODE
****
*/
REC$CO$TBCODE: PROCEDURE;
  IF MSGDATA(0) > 5 THEN DO;
    CALL ERROR(.REC$CO$TBCODE,4,1);
    RETURN;
  END;
  DO CASE MSGDATA(0);
    CALL PRN;
    CALL PRN$PER;
    CALL PRN$DEM;
    CALL PRN$STT$PER;
    CALL PRN$STT$DEM;
    CALL PRN$STT$MSG;
  END;
  RETURN;
END;

```



```

$EJECT
/*
*****
SEND$TASK: SEND TASK DATA TO ANOTHER COMPUTER
*****
*/
SEND$TASK: PROCEDURE (KIND,TASK$N,REC);
  DCL (KIND,TASK$N,REC) BYTE;
  DCL (STEP,MAX) BYTE,
    (ADR$ELE,ADR2) ADDRESS,
    (ELE BASED ADR$ELE) BYTE,
    HEAD(4) BYTE;

  IF KIND = 1 THEN DO ;
    ADR$ELE = .DATA$BLOCK$PER(0).NUMBER;
    STEP = PER$BLOCK$LEN;
    MAX = NUM$PER$TASK;
    ADR2 = .DATA$BLOCK$PER(NUM$PER$TASK-1);
    END;
  ELSE DO;
    ADR$ELE = .DATA$BLOCK$DEM(0).NUMBER;
    STEP = DEM$BLOCK$LEN;
    MAX = NUM$DEM$TASK;
    ADR2 = .DATA$BLOCK$DEM(NUM$DEM$TASK-1);
    END;
  DO B1 = 1 TO MAX;
    IF ELE = TASK$N THEN GO TO DOS;
    ADR$ELE = ADR$ELE + STEP;
    END;
  RETURN;

DOS:   IF KIND = 1 THEN ADR$ELE = ADR$ELE - 4;
        HEAD(0) = REC; HEAD(1) = MOD$ID;
        HEAD(2) = 6; HEAD(3) = STEP + 5;
        IF NOT SEND(.HEAD) THEN RETURN;

```

```
502 2 MSGDATA(0) = KIND;
503 2 CALL MOVE(STEP,ADR$ELE,ADRM$GDATA+1);

504 2 DO A1 = ADR$ELE TO ADR2-1;
505 3 CALL MOVE(STEP,A1+STEP,A1);
506 3 A1 = A1 + STEP;
507 3 END;
508 2 CALL CLEARDATA(ADR2,ADR2+STEP-1);
509 2 IF KIND = 1 THEN NUM$PER$TASK = NUM$PER$TASK - 1;
511 2 ELSE NUM$DEM$TASK = NUM$DEM$TASK - 1;
512 2 RETURN;
513 2 END SEND$TASK;
```



```

$EJECT
/*
*****
REC$TASK: RECEIVES TASK SENDED BY ANOTHER COMPUTER
*****
*/
REC$TASK: PROCEDURE;

    1  IF MSGDATA(0) = 1 THEN DO;
    2      IF NUM$PER$TASK = 8 THEN DO;
    3          CALL ERROR(.REC$TASK,4,2);
    4          RETURN;
    4          END;
    4      CALL MOVE(PER$BLOCK$LEN,ADRM$SGDATA+1,
    3          .DATA$BLOCK$PER(NUM$PER$TASK));
    3          NUM$PER$TASK = NUM$PER$TASK + 1;
    3          RETURN;
    3          END;
    3      ELSE DO;
    2          IF NUM$DEM$TASK = 8 THEN DO;
    3              CALL ERROR(.REC$TASK,4,3);
    4              RETURN;
    4              END;
    4          CALL MOVE(DEM$BLOCK$LEN,ADRM$SGDATA+1,
    3              .DATA$BLOCK$DEM(NUM$DEM$TASK));
    3              NUM$DEM$TASK = NUM$DEM$TASK + 1;
    3              RETURN;
    3              END;
    2      END REC$TASK;
    2

```

```

$EJECT
/*
*****
SET$NEW$SINK: SETS NEW COMPUTER LOCATION FOR ALL MESSAGES
SENDERS OF TASK$N.
*****
*/
SET$NEW$SINK: PROCEDURE (TASK$N,OLD,NEW);
  DCL (TASK$N,OLD,NEW) BYTE;
  DCL ADR$ELE ADDRESS,
      (ELE BASED ADR$ELE) BYTE,
      ELE2 BYTE;
  ADR$ELE = .DATA$BLOCK$PER(0).NUMBER;
  DO B1 = 1 TO NUM$PER$TASK;
    IF ELE = TASK$N THEN DO;
      ADR$ELE = ADR$ELE + 8;
      GO TO DOS;
    END;
    ADR$ELE = ADR$ELE + PER$BLOCK$LEN;
  END;
  ADR$ELE = .DATA$BLOCK$DEM(0).NUMBER;
  DO B1 = 1 TO NUM$DEM$TASK;
    IF ELE = TASK$N THEN DO;
      ADR$ELE = ADR$ELE + 10;
      GO TO DOS;
    END;
    ADR$ELE = ADR$ELE + DEM$BLOCK$LEN;
  END;
RETURN;
DOS:  ELE2 = ELE;
      ADR$ELE = ADR$ELE + 1;
      DO B1 = 1 TO ELE2;
        IF ELE = OLD THEN ELE = NEW;
        ADR$ELE = ADR$ELE + 3;
      END;
RETURN;

```

```
568 1  $EJECT
569 2  /*
571 3  ****
572 3  ****
573 3  ****
574 2  ****
575 3  ****
576 3  ****
577 3  ****
578 2  ****
579 2  ****

    SET$WRITER: SET OUTPUT MODULE DEVICE.
    ****
    */
    SET$WRITER: PROCEDURE;

    IF MSGDATA(0) THEN DO;
        HARD$COPY = TRUE;
        CALL PAR$LINK$REQ(MOD$ID);
        END;
    ELSE DO;
        HARD$COPY = FALSE;
        CALL CON$LINK$REQ(MOD$ID);
        END;

    RETURN;
    END SET$WRITER;
```



```

615      4      CALL SET$WRITER;
616      4      END;
617      3      DO; PRN$FLAG(0), PRN$FLAG(1) = TRUE; /* MN 3 */
619      4      CALL SET$WRITER;
620      4      END;
621      3      DO; PRN$FLAG(2), PRN$FLAG(3), PRN$FLAG(4) = TRUE; /*4*/
623      4      CALL SET$WRITER;
624      4      END;
625      3      CALL SEND$TASK(MSGDATA(0), MSGDATA(1), MSGDATA(2));
626      3      CALL REC$TASK; /* MN 6 */
627      3      CALL SET$NEW$SINK(MSGDATA(0), MSGDATA(1), MSGDATA(2));
628      3      END;
629      2      RETURN;
630      2      END MSGENT4;

```

```

631      1      END EMULA3;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 10E4H      4324D
VARIABLE AREA SIZE = 00CEH      206D
MAXIMUM STACK SIZE = 0012H      18D
1074 LINES READ
0 PROGRAM ERROR(S)

```

```

END OF PL/M-80 COMPILATION

```


ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EMULA4
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY: PLM80 :F1:EMULA4.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)

```

EMULA4: DO;
/*
*****
***** THIS PROGRAM CONTAINS PROCEDURES FOR THE INTERACTION WITH
        THE USER.
*****
**/
$NOLIST
$ INCLUDE (:F1: EMUMSG.ONE)
/*
*****
***** : EMULA4 LOCAL DATA. IT CONTAINS THE MESSAGES AND CONTROL VARIABLES
        FOR THE INTERACTION WITH THE USER.
*****
**/

DCL STATE BYTE PUBLIC , /* STATE OF INPUT PROCESS */
WRITING   BYTE PUBLIC , /* ENTRY STATE FLAG */
COUNTER   BYTE PUBLIC , /* INDEX FOR COUNTER PROCEDURE */
EMULATION$TIME ADDRESS PUBLIC , /* TIME LIMIT FOR EMULATION */
EMU$CLOCK1(4) BYTE PUBLIC , /* CLOCK SAVER NUMBER 1 */
EMU$CLOCK2(4) BYTE PUBLIC , /* CLOCK SAVER NUMBER 2 */
EMU$CLOCK3(4) BYTE PUBLIC , /* CLOCK SAVER NUMBER 3 */
DATA$SET$MSG(4) BYTE PUBLIC , /* HEADER OF NETWORK MSG DATA */
OUT$TIME(14) BYTE PUBLIC , /* OUTPUT TIME MESSAGE */
READY     BYTE PUBLIC , /* READY FOR EMULATION (FLAG) */
NUM$CPTR$USED BYTE PUBLIC , /* NUMBER OF COMPUTERS REQUESTED */
CPTR$IN$USE  BYTE PUBLIC , /* CURRENT COMPUTER IN USE */

```



```

= = = = =
TASK$NUMBER      BYTE PUBLIC,/* NUMBER OF TASK INFO RECEIVED */
MSG$NUM          BYTE PUBLIC,/* # OF MSG DATA RECEIVED*/
PER$TASK$NUM     BYTE PUBLIC,/* # OF PERIODIC TASK DATA REC. */
DEM$TASK$NUM     BYTE PUBLIC,/* # OF DEMAND TASK DATA REC.*/
NPT              BYTE PUBLIC,/* # PERIODIC TASK TO SIMULATE */
NMO              BYTE PUBLIC,/* NUMBER OF MSG TO BE SENDED */
MO               BYTE PUBLIC,/* NUMBER OF MSG PROCESSED*/
NDT              BYTE PUBLIC,/* # OF DEMAND TASK TO SIMULATE*/
NMI              BYTE PUBLIC,/* NUMBER OF MSG TO BE RECEIVED */
MI               BYTE PUBLIC;/* NUMBER OF MSG RECIED ALREADY*/

= = = = =
100 1  DECLARE /* THE FOLLOWING MESSAGES TEXTS */
MSG01(*) BYTE DATA(0, 'NUMBER OF COMPUTERS NEEDED.?(<=3) .'),
MSG02(*) BYTE DATA(1, 'CONSOLE CONNECTION REFUSED III'),
MSG03(*) BYTE DATA(1, 'CONSOL RELEASED III'),
MSG05(*) BYTE DATA(1, 'DATA LOADING PROCESS ...'),
MSG06(*) BYTE DATA(1, 'END DATA LOADING PROCESS ...'),
MSG07(*) BYTE DATA(0, 'LOADING COMPUTER ...'),
MSG08(*) BYTE DATA(1, 'PERIODIC TASK DATA...:'),
MSG09(*) BYTE DATA(0, 'NUMBER OF PERIODIC TASK (<=8) .? .'),
MSG11(*) BYTE DATA(1, 'DEMAND TASK DATA...:'),
MSG12(*) BYTE DATA(0, 'PERIODIC TASK ...'),
MSG13(*) BYTE DATA(1, 'END PERIODIC TASK. '),
MSG14(*) BYTE DATA(0, 'ID NUMBER (<256) .?'),
MSG15(*) BYTE DATA(1, 'ALREADY USED III'),
MSG16(*) BYTE DATA(0, 'PERIOD (IN MS) .?'),
MSG17(*) BYTE DATA(1, 'WRONG DATA IIIIII'),
MSG18(*) BYTE DATA(0, 'COMPUTATIONAL DELAY (IN MS) .? .'),
MSG19(*) BYTE DATA(0, 'NUMBER OF OUTPUT MSGS (<=4) .? .'),
MSG20(*) BYTE DATA(1, 'END OUTPUT MSGS'),
MSG21(*) BYTE DATA(0, 'OUTPUT MESSAGE (##,##) ? .'),
MSG22(*) BYTE DATA(1, ' '),
MSG23(*) BYTE DATA(0, 'NUMBER OF DEMAND TASK (<=8) .?'),
MSG24(*) BYTE DATA(1, 'END DATA FOR THIS COMPUTERII'),
MSG25(*) BYTE DATA(1, 'END DEMAND TASK DATA'),
MSG26(*) BYTE DATA(0, 'DEMAND TASK....'),

```

302

303

```

$EJECT
/*
*****
GET$NUMBER: CHECK NUMBER SENT BY INPUT MODULE
*****
*/
111 1 GET$NUMBER: PROCEDURE (ADR,LEN) BYTE ;
112 2 DCL ADR ADDRESS, LEN BYTE;
113 2 IF NOT GETNUM THEN DO;
115 3 CALL PRINT$SYNC(.MSG17,LENGTH(MSG17),EM);
116 3 CALL PRINT$SYNC(ADR,LEN,EM);
117 3 CALL CON$INPUT$REQ(EM,TRUE);
118 3 RETURN FALSE;
119 3 END;
120 2 RETURN TRUE;
121 2 END GET$NUMBER;
/*
*****
SEND$INFO: SEND INFORMATION TO EMULATOR
*****
*/
122 1 SEND$INFO: PROCEDURE (MGN,MGL,EADR,E1,E2) BYTE PUBLIC;
123 2 DCL EADR ADDRESS,(MGN,MGL,E1,E2) BYTE;
124 2 DATA$SET$MSG(MN) = MGN; /* SET MSG NUMBER */
125 2 DATA$SET$MSG(ML) = MGL; /* SET MSG LENGTH */
126 2 IF NOT SEND(.DATA$SET$MSG) THEN DO;
128 3 CALL ERROR(EADR,E1,E2);
129 3 RETURN FALSE;
130 3 END;
131 2 RETURN TRUE;
132 2 END SEND$INFO;

```



```

$EJECT
/*
*****
SAVE$DELAY: SAVES INFORMATION ABOUT TASK DELAY AND SENDS IT TO
CORRESPONDING COMPUTER.
*****
*/
155 1 SAVE$DELAY: PROCEDURE (P1,P2,P3,P4) BYTE;
156 2 DCL (P1,P3,P4) BYTE, P2 ADDRESS;

157 2 IF NOT GET$NUMBER(.MSG18,LENGTH(MSG18)) THEN RETURN FALSE;
159 2 IF NOT SEND$INFO(P1,6,P2,6,P3) THEN RETURN FALSE;
161 2 MSGDATA(0) = P4 - 1;
162 2 MSGDATA(1) = BL;
163 2 RETURN TRUE;
164 2 END SAVE$DELAY;
/*
*****
CHECK$MSG$ID: CHECK IF MSG NUMBER HAS BEEN USED BEFORE
*****
*/
165 1 CHECK$MSG$ID: PROCEDURE(ID) BYTE;
166 2 DCL ID BYTE;

167 2 IF MSG$NUM = 0 THEN RETURN TRUE;
169 2 DO B2 = 0 TO MSG$NUM - 1;
170 3 IF MSG$TBL (B2).NUMBER = ID THEN RETURN FALSE;
172 3 END;
173 2 RETURN TRUE;
174 2 END CHECK$MSG$ID;

```



```

175 $EJECT
176 /*
177 *****
178 *****
179 *****
180 *****
181 *****
182 *****
183 *****
184 *****
185 *****
186 *****
187 *****
188 *****
189 *****
190 *****
191 *****
192 *****
193 *****
194 *****
195 *****
196 *****
197 *****
198 *****

175 CHECK$TASK$ID: CHECK IF TASK ID HAS BEEN USED BEFORE
176 /*
177 *****
178 *****
179 *****
180 *****
181 *****
182 *****
183 *****
184 *****
185 *****
186 *****
187 *****
188 *****
189 *****
190 *****
191 *****
192 *****
193 *****
194 *****
195 *****
196 *****
197 *****
198 *****

175 CHECK$TASK$ID: PROCEDURE(ID) BYTE;
176 DCL ID BYTE;
177
178 IF ID = 0 THEN RETURN FALSE;
179 DO B2 = 0 TO TASK$NUMBER - 1;
180 IF ID$TBL(B2).NUMBER = ID THEN RETURN FALSE;
181 END;
182 RETURN TRUE;
183 END CHECK$TASK$ID;
184 /*
185 *****
186 *****
187 *****
188 *****
189 *****
190 *****
191 *****
192 *****
193 *****
194 *****
195 *****
196 *****
197 *****
198 *****

175 PRN$SEQUENCE: PRINT SEQUENCE NUMBER ON CONSOL
176 /*
177 *****
178 *****
179 *****
180 *****
181 *****
182 *****
183 *****
184 *****
185 *****
186 *****
187 *****
188 *****
189 *****
190 *****
191 *****
192 *****
193 *****
194 *****
195 *****
196 *****
197 *****
198 *****

175 PRN$SEQUENCE: PROCEDURE (P1);
176 DCL P1 BYTE;
177 DO CASE P1;
178 CALL PRINT$SYNC(.MSG79,LENGTH(MSG79),EM);
179 CALL PRINT$SYNC(.MSG71,LENGTH(MSG71),EM);
180 CALL PRINT$SYNC(.MSG72,LENGTH(MSG72),EM);
181 CALL PRINT$SYNC(.MSG73,LENGTH(MSG73),EM);
182 CALL PRINT$SYNC(.MSG74,LENGTH(MSG74),EM);
183 CALL PRINT$SYNC(.MSG75,LENGTH(MSG75),EM);
184 CALL PRINT$SYNC(.MSG76,LENGTH(MSG76),EM);
185 CALL PRINT$SYNC(.MSG77,LENGTH(MSG77),EM);
186 CALL PRINT$SYNC(.MSG78,LENGTH(MSG78),EM);
187 END;
188 RETURN;
189

```

```
$EJECT
/*
```

```
*****
```

```
CHECK$LESS$THAN: CHECK UPPER LIMIT BOUND
```

```
*****
```

```
*/
```

```
CHECK$LESS$THAN: PROCEDURE (P1,P2,P3,P4) BYTE;
```

```
DCL (P1,P3,P4) BYTE, P2 ADDRESS;
```

```
IF BL > P1 THEN DO;
```

```
CALL PRINT$SYNC(.MSG17,LENGTH(MSG17),EM);
```

```
CALL QUEST(P2,P3,P4);
```

```
RETURN FALSE;
```

```
END;
```

```
RETURN TRUE;
```

```
END CHECK$LESS$THAN;
```

```
/*
```

```
*****
```

```
SLAVE1: WORKS FOR STATES 10 AND 18
```

```
*****
```

```
*/
```

```
SLAVE1: PROCEDURE (P1,P2);
```

```
DCL (P1,P2) BYTE;
```

```
MO = MO + 1;
```

```
IF MO > NMO THEN DO;
```

```
CALL PRINT$SYNC(.MSG20,LENGTH(MSG20),EM);
```

```
CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,P1);
```

```
RETURN;
```

```
END;
```

```
CALL PRINT$SYNC(.MSG21,LENGTH(MSG21),EM);
```

```
CALL PRN$SEQUENCE(MO);
```

```
CALL CON$INPUT$REQ(EM,TRUE);
```

```
STATE = P2;
```

```
RETURN;
```

```
200 1
201 2
202 2
204 3
205 3
206 3
207 3
208 2
209 2
```

```
210 1
211 2
212 2
213 2
215 3
216 3
217 3
218 3
219 2
220 2
221 2
222 2
223 2
```

```

225 1  $EJECT
226 2  /*
227 2  ****
228 2  *****
229 2  *****
230 2  *****
231 2  *****
232 2  *****
233 2  *****
234 2  *****
235 2  *****
236 2  *****
237 2  *****
238 2  *****
239 2  *****
240 2  *****
241 2  *****
242 2  *****
243 2  *****
244 2  *****
245 2  *****
246 2  *****
247 2  *****
248 2  *****
249 2  *****
250 2  *****
251 2  *****
252 2  *****
253 2  *****

SLAVE2: WORKS FOR STATES 11 AND 19.
*/
SLAVE2: PROCEDURE (P1,P2);
  DCL (P1,P2) BYTE;
  DCL (RTSK,RMGN,MGLN) BYTE;

  IF NOT GET$NUMBER(.MSG21,LENGTH(MSG21)) THEN RETURN;
  RTSK = BL;
  IF NOT GET$NUMBER(.MSG21,LENGTH(MSG21)) THEN RETURN;
  RMGN = BL;
  IF NOT CHECK$MSG$ID(RMGN) THEN DO;
    CALL PRINT$SYNC(.MSG15,LENGTH(MSG15),EM);
    CALL QUEST(.MSG21,LENGTH(MSG21),P1);
    RETURN;
  END;
  IF NOT GET$NUMBER(.MSG21,LENGTH(MSG21)) THEN RETURN;
  IF NOT CHECK$LESS$THAN(100,.MSG21,LENGTH(MSG21),P1) THEN RETURN;
  MGLN = BL;
  MSG$TBL(MSG$NUM).NUMBER = RMGN;
  MSG$TBL(MSG$NUM).INDEX = SHL(MO-1,2);
  MSG$TBL(MSG$NUM).SOURCE = ID$TBL(TASK$NUMBER-1).NUMBER;
  MSG$TBL(MSG$NUM).SINK = RTSK;
  MSG$TBL(MSG$NUM).LENT = MGLN;
  MSG$NUM = MSG$NUM + 1;
  CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,P2);
  RETURN;
END SLAVE2;

```

THE UNIVERSITY OF CHICAGO PRESS

THE FOLLOWING PROCEDURES CARRY ON THE INTERFAS WITH THE USER
IN SETTING DOWN THE EMULATION PARAMETERS.

```

2700 1
2701 2
2702 2
2703 3
2704 3
2705 3
2706 3
2707 2
2708 2
2709 2
2710 2
2711 2
2712 2
2713 2
2714 2
2715 2
2716 2
2717 2
2718 2
2719 2
2720 2
2721 2
2722 2
2723 2
2724 3
2725 3
2726 3
2727 2
2728 2
2729 2
2730 2
2731 2
2732 2
2733 2
2734 2
2735 2
2736 2
2737 2
2738 2
2739 2
2740 2
2741 2
2742 2
2743 2
2744 2
2745 2
2746 2
2747 2
2748 2
2749 2
2750 2
2751 2
2752 2
2753 2
2754 2
2755 2
2756 2
2757 2
2758 2
2759 2
2760 2
2761 2
2762 2
2763 2
2764 2
2765 2
2766 2
2767 2
2768 2
2769 2
2770 2
2771 2
2772 2
2773 2
2774 2
2775 2
2776 2
2777 2
2778 2
2779 2
2780 2
2781 2
2782 2
2783 2
2784 2
2785 2
2786 2
2787 2
2788 2
2789 2
2790 2
2791 2
2792 2
2793 2
2794 2
2795 2
2796 2
2797 2
2798 2
2799 2
2800 2
2801 2
2802 2
2803 2
2804 2
2805 2
2806 2
2807 2
2808 2
2809 2
2810 2
2811 2
2812 2
2813 2
2814 2
2815 2
2816 2
2817 2
2818 2
2819 2
2820 2
2821 2
2822 2
2823 2
2824 2
2825 2
2826 2
2827 2
2828 2
2829 2
2830 2
2831 2
2832 2
2833 2
2834 2
2835 2
2836 2
2837 2
2838 2
2839 2
2840 2
2841 2
2842 2
2843 2
2844 2
2845 2
2846 2
2847 2
2848 2
2849 2
2850 2
2851 2
2852 2
2853 2
2854 2
2855 2
2856 2
2857 2
2858 2
2859 2
2860 2
2861 2
2862 2
2863 2
2864 2
2865 2
2866 2
2867 2
2868 2
2869 2
2870 2
2871 2
2872 2
2873 2
2874 2
2875 2
2876 2
2877 2
2878 2
2879 2
2880 2
2881 2
2882 2
2883 2
2884 2
2885 2
2886 2
2887 2
2888 2
2889 2
2890 2
2891 2
2892 2
2893 2
2894 2
2895 2
2896 2
2897 2
2898 2
2899 2
2900 2
2901 2
2902 2
2903 2
2904 2
2905 2
2906 2
2907 2
2908 2
2909 2
2910 2
2911 2
2912 2
2913 2
2914 2
2915 2
2916 2
2917 2
2918 2
2919 2
2920 2
2921 2
2922 2
2923 2
2924 2
2925 2
2926 2
2927 2
2928 2
2929 2
2930 2
2931 2
2932 2
2933 2
2934 2
2935 2
2936 2
2937 2
2938 2
2939 2
2940 2
2941 2
2942 2
2943 2
2944 2
2945 2
2946 2
2947 2
2948 2
2949 2
2950 2
2951 2
2952 2
2953 2
2954 2
2955 2
2956 2
2957 2
2958 2
2959 2
2960 2
2961 2
2962 2
2963 2
2964 2
2965 2
2966 2
2967 2
2968 2
2969 2
2970 2
2971 2
2972 2
2973 2
2974 2
2975 2
2976 2
2977 2
2978 2
2979 2
2980 2
2981 2
2982 2
2983 2
2984 2
2985 2
2986 2
2987 2
2988 2
2989 2
2990 2
2991 2
2992 2
2993 2
2994 2
2995 2
2996 2
2997 2
2998 2
2999 2
3000 2

```



```

278 3      CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
279 3      CALL PRINT$SYNC(.MSG07,LENGTH(MSG07),EM);
280 3      CALL PRN$SEQUENCE(CPTR$IN$USE);
281 3      CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,2);
282 3      END;
283 2      RETURN;
284 2      END STATE01;
      /*****/
285 1      STATE02: PROCEDURE;
286 2      DATA$SET$MSG (RMN) = SHL(CPTR$IN$USE-1,3) + 2 ;
287 2      DATA$SET$MSG (SMN) = EM;
288 2      IF NOT SEND$INFO(15,4,.,STATE02,6,2) THEN RETURN;
289 2      CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
290 2      CALL PRINT$LINKED(.MSG08,LENGTH(MSG08),EM,3);
291 2      RETURN;
292 2      END STATE02;
293 2      /*****/
294 1      STATE03: PROCEDURE;
295 2      CALL QUEST(.MSG09,LENGTH(MSG09),4);
296 2      RETURN;
297 2      END STATE03;
      /*****/
298 1      STATE04: PROCEDURE;
299 2      IF NOT GET$NUMBER(.MSG09,LENGTH(MSG09)) THEN RETURN;
300 2      IF NOT CHECK$LESS$THAN(8,.MSG09,LENGTH(MSG09),4) THEN RETURN;
301 2      IF BL = 0 THEN DO;
302 2          CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
303 2          CALL PRINT$LINKED(.MSG11,LENGTH(MSG11),EM,12);
304 2          RETURN;
305 2      END;
306 2      IF NOT SEND$INFO(16,5,.,STATE04,6,4) THEN RETURN;
307 2      MSGDATA(0) = BL;
308 2      PER$TASK$NUM = 0 ; NPT = BL;
309 2      CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,05);
310 2      RETURN;
311 2      END STATE04;
312 2
313 2
314 2
315 2
316 2

```

```

317 1 317 *****/
318 2 STATE05: PROCEDURE;
319 2 PER$TASK$NUM = PER$TASK$NUM + 1;
320 2 IF PER$TASK$NUM > NPT THEN DO;
321 3 CALL PRINT$SYNC(.MSG13,LENGTH(MSG13),EM);
322 3 CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
323 3 CALL PRINT$LINKED(.MSG11,LENGTH(MSG11),EM,12);
324 3 RETURN;
325 3 END;
326 2 CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
327 2 CALL PRINT$SYNC(.MSG12,LENGTH(MSG12),EM);
328 2 CALL PRN$SEQUENCE(PER$TASK$NUM);
329 2 CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
330 2 CALL QUEST(.MSG14,LENGTH(MSG14),06);
331 2 RETURN;
332 2 END STATE05;
333 2 *****/
334 1 STATE06: PROCEDURE;
335 2 IF NOT GET$NUMBER(.MSG14,LENGTH(MSG14)) THEN RETURN;
336 2 IF NOT CHECK$TASK$ID(BL) THEN DO;
337 3 CALL PRINT$SYNC(.MSG15,LENGTH(MSG15),EM);
338 3 CALL QUEST(.MSG14,LENGTH(MSG14),06);
339 3 RETURN;
340 3 END;
341 3 END;
342 2 IF NOT SAVE$TASK(PER$TASK$NUM,1,2,.STATE06,6) THEN RETURN;
343 2 CALL QUEST(.MSG16,LENGTH(MSG16),7);
344 2 RETURN;
345 2 END STATE06;
346 2 *****/
347 1 STATE07: PROCEDURE;
348 2 IF NOT GET$NUMBER(.MSG16,LENGTH(MSG16)) THEN RETURN;
349 2 IF NOT SEND$INFO(4,7,.STATE07,6,7) THEN RETURN;
350 2 MSGDATA(0) = PER$TASK$NUM - 1;
351 2 MSGDATA(1) = BU;
352 2 MSGDATA(2) = BL;
353 2 CALL QUEST(.MSG18,LENGTH(MSG18),8);
354 2
355 2

```



```

356 2      RETURN;
357 2      END STATE07;
      /*****/
358 1      STATE08: PROCEDURE;
359 2      IF NOT SAVE$DELAY(3,.STATE08,8,PER$TASK$NUM) THEN RETURN;
361 2      CALL QUEST(.MSG19,LENGTH(MSG19),9);
362 2      RETURN;
363 2      END STATE08;
      /*****/
364 1      STATE09: PROCEDURE;
365 2      IF NOT GET$NUMBER(.MSG19,LENGTH(MSG19)) THEN RETURN;
367 2      IF NOT CHECK$LESS$THAN(4,.MSG19,LENGTH(MSG19),9) THEN RETURN;
369 2      IF NOT SEND$INFO(5,6,.STATE09,6,9) THEN RETURN;
371 2      MSGDATA(0) = PER$TASK$NUM - 1;
372 2      MSGDATA(1) = BL;
373 2      NMO = BL;
374 2      MO = 0;
375 2      CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,10);
376 2      RETURN;
377 2      END STATE09;
      /*****/
378 1      STATE10: PROCEDURE;
379 2      CALL SLAVE1(5,11);
380 2      RETURN;
381 2      END STATE10;
      /*****/
382 1      STATE11: PROCEDURE;
383 2      CALL SLAVE2(11,10);
384 2      RETURN;
385 2      END STATE11;
      /*****/
386 1      STATE12: PROCEDURE;
387 2      CALL QUEST(.MSG23,LENGTH(MSG23),13);
388 2      RETURN;
389 2      END STATE12;
      /*****/

```

```

390 1 STATE13: PROCEDURE;
391 2 IF NOT GET$NUMBER(.MSG23,LENGTH(MSG23)) THEN RETURN;
393 2 IF NOT CHECK$LESS$THAN(8,.MSG23,LENGTH(MSG23),13) THEN RETURN;
395 2 IF BL = 0 THEN DO;
397 3 CALL PRINT$LINKED(.MSG24,LENGTH(MSG24),EM,01);
398 3 RETURN;
399 3 END;
400 2 IF NOT SEND$INFO(1,5,.STATE13,6,13) THEN RETURN;
402 2 MSGDATA(0) = BL;
403 2 DEM$TASK$NUM = 0;
404 2 NDT = BL;
405 2 CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,14);
406 2 RETURN;
407 2 END STATE13;
    /*****/
408 1 STATE14: PROCEDURE;
409 2 DEM$TASK$NUM = DEM$TASK$NUM + 1;
410 2 IF DEM$TASK$NUM > NDT THEN DO;
412 3 CALL PRINT$SYNC(.MSG25,LENGTH(MSG25),EM);
413 3 CALL PRINT$LINKED(.MSG24,LENGTH(MSG24),EM,01);
414 3 RETURN;
415 3 END;
416 2 CALL PRINT$SYNC(.MSG26,LENGTH(MSG26),EM);
417 2 CALL PRN$SEQUENCE(DEM$TASK$NUM);
418 2 CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
419 2 CALL QUEST(.MSG14,LENGTH(MSG14),15);
420 2 RETURN;
421 2 END STATE14;
    /*****/
422 1 STATE15: PROCEDURE;
423 2 IF NOT GET$NUMBER(.MSG14,LENGTH(MSG14)) THEN RETURN;
425 2 IF NOT CHECK$TASK$ID(BL) THEN DO;
427 3 CALL PRINT$SYNC(.MSG15,LENGTH(MSG15),EM);
428 3 CALL QUEST(.MSG14,LENGTH(MSG14),15);
429 3 RETURN;
430 3 END;

```

```

431 2 IF NOT SAVE$TASK(DEM$TASK$NUM,2,7,.STATE15,15) THEN RETURN;
433 2 CALL QUEST(.MSG18,LENGTH(MSG18),16);
434 2 RETURN;
435 2 END STATE15;
    /****/
436 1 STATE16: PROCEDURE;
437 2 IF NOT SAVE$DELAY(8,.STATE16,16,DEM$TASK$NUM) THEN RETURN;
439 2 CALL QUEST(.MSG19,LENGTH(MSG19),17);
440 2 RETURN;
441 2 END STATE16;
    /****/
442 1 STATE17: PROCEDURE;
443 2 IF NOT GET$NUMBER(.MSG19,LENGTH(MSG19)) THEN RETURN;
445 2 IF NOT CHECK$LESS$THAN(4,.MSG19,LENGTH(MSG19),17) THEN RETURN;
447 2 IF NOT SEND$INFO(10,6,.STATE17,6,17) THEN RETURN;
449 2 MSGDATA(0) = DEM$TASK$NUM -1;
450 2 MSGDATA(1) = BL;
451 2 NMO = BL;
452 2 MO = 0;
453 2 CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,18);
454 2 RETURN;
455 2 END STATE17;
    /****/
456 1 STATE18: PROCEDURE;
457 2 CALL SLAVE1(20,19);
458 2 RETURN;
459 2 END STATE18;
    /****/
460 1 STATE19: PROCEDURE;
461 2 CALL SLAVE2(19,18);
462 2 RETURN;
463 2 END STATE19;
    /****/
464 1 STATE20: PROCEDURE;
465 2 CALL QUEST(.MSG27,LENGTH(MSG27),21);
466 2 RETURN;

```

```

467      2      END STATE20;
          /****/
468      1      STATE21: PROCEDURE;
469      2      IF NOT GET$NUMBER(.MSG27,LENGTH(MSG27)) THEN RETURN;
471      2      IF NOT CHECK$LESS$THAN(4,.MSG27,LENGTH(MSG27),21) THEN RETURN;
473      2      IF NOT SEND$INFO(12,6,.STATE21,6,21) THEN RETURN;
475      2      MSGDATA(0) = DEM$TASK$NUM - 1;
476      2      MSGDATA(1) = BL;
477      2      NMI = BL;
478      2      MI = 0;
479      2      CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,22);
480      2      RETURN;
481      2      END STATE21;
          /****/
482      1      STATE22: PROCEDURE;
483      2      MI = MI + 1;
484      2      IF MI > NMI THEN DO;
486      3      CALL PRINT$LINKED(.MSG28,LENGTH(MSG28),EM,24);
487      3      RETURN;
488      3      END;
489      2      CALL PRINT$SYNC(.MSG29,LENGTH(MSG29),EM);
490      2      CALL PRN$SEQUENCE(MI);
491      2      CALL CON$INPUT$REQ(EM,TRUE);
492      2      STATE = 23;
493      2      RETURN;
494      2      END STATE22;
          /****/
495      1      STATE23: PROCEDURE;
496      2      IF NOT GET$NUMBER(.MSG29,LENGTH(MSG29)) THEN RETURN;
498      2      IF NOT SEND$INFO(13,7,.STATE23,6,23) THEN RETURN;
500      2      MSGDATA(0) = DEM$TASK$NUM - 1;
501      2      MSGDATA(1) = SHL(MI-1,2) + SHL(MI-1,1);
502      2      MSGDATA(2) = BL;
503      2      CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,22);
504      2      RETURN;
505      2      END STATE23;

```

317


```

540 1 STATE29: PROCEDURE;
541 2   CALL PRINT$SYNC(.MSG34,LENGTH(MSG34),EM);
542 2   CALL PRINT$SYNC(.MSG35,LENGTH(MSG35),EM);
543 2   CALL PRINT$LINKED(.MSG36,LENGTH(MSG36),EM,30);
544 2   RETURN;
545 2   END STATE29;
      /*****/
546 1 STATE30: PROCEDURE;
547 2   CALL PRINT$SYNC(.MSG37,LENGTH(MSG37),EM);
548 2   CALL QUEST(.MSG38,LENGTH(MSG38),31);
549 2   RETURN;
550 2   END STATE30;
      /*****/
551 1 STATE31: PROCEDURE;
552 2   DCL EOT LIT '04H';
553 2   DCL COMM$LIST(*) BYTE DATA('WEMS',EOT);
554 2   DO B2 = 0 TO LAST(COMM$LIST);
555 3     IF MSGDATA(B1) = COMM$LIST(B2) THEN GO TO FOUND;
556 3   END;
557 3   CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
558 2   RETURN;
559 2   FOUND: B1 = B1 + 1;
560 2   DO CASE B2;
561 3     CALL WRITE$DATA;
562 3     CALL START$EMU;
563 3     CALL MOVE$TASK;
564 3     CALL SUBS$DATA;
565 3     CALL CON$RELEASE;
566 3   END;
567 3   RETURN;
568 2   END STATE31;
      /*****/
569 1 STATE32: PROCEDURE;
570 2   IF NOT GET$CHAR THEN DO;
571 3     CALL QUEST(.MSG80,LENGTH(MSG80),32);
572 3   RETURN;
573 3
574 3

```



```
575 3      END;  
576 2      IF BL <> 'Y' THEN DO;  
578 3          CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,29);  
579 3          RETURN;  
580 3      END;  
581 2      CALL QUEST(.MSG01,LENGTH(MSG01),00);  
582 2      RETURN;  
583 2      END STATE32;
```

```

584 1  $EJECT
585 2  /**
587 2  ****
589 3  ****
590 3  ****
591 3  ****
592 3  ****
593 2  ****
595 3  ****
596 3  ****
597 3  ****
598 2  ****
599 3  ****
600 3  ****
601 3  ****
602 3  ****
603 3  ****
604 3  ****
605 3  ****
606 3  ****
607 3  ****
608 3  ****
609 3  ****
610 3  ****
611 3  ****
612 3  ****
613 3  ****
614 3  ****

RECEIVE$INPUT: RECEIVE INPUT FROM SYSTEM CONSOLE
**/
RECEIVE$INPUT: PROCEDURE PUBLIC;

    B1 = 0; CALL DEBLK;
    IF MSGDATA(B1) = '%' THEN DO;
        STATE = 29;
        CALL STATE29;
        RETURN;
    END;
    IF STATE > 32 THEN DO;
        CALL ERROR(.RECEIVE$INPUT,3,1);
        RETURN;
    END;
    DO CASE STATE;
        CALL STATE00;
        CALL STATE01;
        CALL STATE02;
        CALL STATE03;
        CALL STATE04;
        CALL STATE05;
        CALL STATE06;
        CALL STATE07;
        CALL STATE08;
        CALL STATE09;
        CALL STATE10;
        CALL STATE11;
        CALL STATE12;
        CALL STATE13;
        CALL STATE14;
        CALL STATE15;
    
```

615	3	CALL STATE16;
616	3	CALL STATE17;
617	3	CALL STATE18;
618	3	CALL STATE19;
619	3	CALL STATE20;
620	3	CALL STATE21;
621	3	CALL STATE22;
622	3	CALL STATE23;
623	3	CALL STATE24;
624	3	CALL STATE25;
625	3	CALL STATE26;
626	3	CALL STATE27;
627	3	CALL STATE28;
628	3	CALL STATE29;
629	3	CALL STATE30;
630	3	CALL STATE31;
631	3	CALL STATE32;
632	3	END;
633	2	END RECEIVE\$INPUT;

```

$EJECT
/*

```

```

*****

```

```

WRITE$ACK$RECEIVED: WRITE ACKNOWLEDGE RECEIVED FROM SYSTEM CONSOLE
*****

```

```

*/

```

```

WRITE$ACK$RECEIVED: PROCEDURE PUBLIC;

```

```

    STATE = MSGDATA(0);

```

```

    CALL RECEIVE$INPUT;

```

```

    RETURN;

```

```

    END WRITE$ACK$RECEIVED;

```

```

    END EMULA4;

```

```

634 1
635 2
636 2
637 2
638 2
639 1

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 11A8H 4520D
VARIABLE AREA SIZE = 0057H 87D
MAXIMUM STACK SIZE = 000EH 14D
985 LINES READ
0 PROGRAM ERROR(S)

```

```

END OF PL/M-80 COMPILATION

```

LIST OF REFERENCES

- [1] Arnold, C.E., "The need for distributed operating systems", Naval Underwater Systems Center, New London Laboratory, New London, Connecticut 06320, 15 april 1975.
- [2] Niemann, W., "A Real-Time Operating System for Single Board Computer Based Distributed Naval Tactical Data Systems", M.S. Thesis, Naval Postgraduate School, Monterey, June 1978.
- [3] Naval Postgraduate School, "A study of alternatives for VSTOL computer systems", by U. Kodres, J. Butlinger, R. Hamming and C. Jones, April 1978.
- [4] Breuer, M.A. Editor, "Design Automation of Digital Systems: Theory and Techniques", Prentice Hall, 1972.
- [5] "SBC 80/20 and SBC 80/20-4 Single Board Computer Hardware Reference Manual", Intel Corporation, 1977.
- [6] INTELLEC Microcomputer Development System, "Hardware Reference Manual", Intel Corporation, November 1976.
- [7] ISIS-II User's Guide, Intel Corporation, 1978.
- [8] ISIS-II PL/M-80 Compiler Operator's Manual, Intel Corporation, 1977.

INITIAL DISTRIBUTION LIST

	No. copies
1. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
2. Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
3. Assoc. Prof. Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4. LTC Roger R. Schell, Code 52Sj Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5. LT(JG) Luis A. Guillen, Peruvian Navy Central de Procesamiento de Datos Ministerio de Marina Salaverry S/N, San Felipe Lima, Peru	1
6. LT(JG) Javier De la Cuba, Peruvian Navy Central de Procesamiento de Datos Ministerio de Marina Salaverry S/N, San Felipe Lima, Peru	1
7. Direccion de Instruccion de la Marina Ministerio de Marina Salaverry S/N, San Felipe Lima, Peru	1
8. LT Felix Luna, Peruvian Navy 1149 Leahy Rd. Monterey, California 93940	1
9. LCDR. Amrun Sehan 415 Casaverde #3 Monterey, California 93940	1
10. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2

